MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

AI Memo No. 531 July, 1979

An Overview of A Theory of Syntactic Recognition for Natural Language

Mitchell P. Marcus

Assume that the syntax of natural language can be parsed by a left-to-right deterministic mechanism without facilities for parallelism or backup. It will be shown that this "determinism" hypothesis, explored within the context of the grammar of English, leads to a simple mechanism, a grammar interpreter, having the following properties:

(a) Simple rules of grammar can be written for this interpreter which capture the generalizations behind various linguistic phenomena, despite the seeming difficulty of capturing such generalizations in the framework of a processing model for recognition.

(b) The structure of the grammar interpreter constrains its operation in such a way that grammar rules cannot parse sentences which violate either of two constraints which Chomsky claims are linguistic universals. This result depends in part upon the computational use of Chomsky's notion of Annotated Surface Structure.

(c) The grammar interpreter provides a simple explanation for the difficulty caused by "garden path" sentences, such as "The cotton clothing is made of grows in Mississippi".

To the extent that these properties, all of which reflect deep properties of natural language, follow from the original hypothesis, they provide indirect evidence for the truth of this assumption.

This memo is an abridged form of several topics discussed at length in [Marcus 77]; it does *not* discuss the mechanism used to parse noun phrases nor the kinds of interaction between syntax and semantics discussed in that work.

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-75-C-0643 and in part by the Office of Naval Research under Office of Naval Research contract N00014-77-C-0389.

Table of Contents

I. The Determinism Hypothesis

Introduction

All current natural language parsers that are adequate to cover a wide range of syntactic constructions operate by simulating nondeterministic machines, either by using backtracking or by pseudo-parallelism. On the face of it, this seems to be necessary, for a cursory examination of natural language reveals many phenomena that seem to demand nondeterministic solutions if we restrict our attention to parsers that operate left-to-right.

A typical natural language parser is conceptually nondeterministic in that it will parse a given input if there is some sequence of grammar rules (however such rules are expressed) whose application yields a coherent analysis of the input, even if other legal sequences of rule application do not lead to such analyses. Since all physically existing machines must be deterministic, such a nondeterministic machine must be simulated by causing a deterministic machine to make "guesses" about what the proper sequence of actions for a given input should be, coupled with some mechanism for aborting incorrect guesses. For many inputs, this necessarily leads to the creation of some syntactic substructures which are not constituents in whatever final syntactic analysis is assigned to a given input.

To see that such an approach seems to be necessary, consider the sentences 1.1a and 1.2a shown below. While the first seven words of both of these sentences are identical, their structures, as shown as 1.1b and 1.2b, are very different. In 1.2a "have" is the main verb of an imperative sentence, with the rest of the sentence a subordinate clause. 1.2b is a question, with "have" as an auxiliary of the main verb "taken", and "the students" as the subject of the main clause.

1.1a Have the students who missed the exam take the makeup today.
1.1b [$_S$ [$_{VP}$ Have [$_S$ the students who ... take the makeup today]]]

1.2a Have the students who missed the exam taken the makeup today?
1.2b [$_S$ [$_{AUX}$ Have] [$_{NP}$ the students who ...] [$_{VP}$ taken the exam today]]

Figure 1 - Some examples that motivate nondeterministic parsing.

It would seem that to analyze the structure of these sentences in a left-to-right manner, a parser must necessarily simulate a nondeterministic process. Not only is it impossible to determine what role the word "have" serves in either 1.1a or 1.2a upon first encounter, but the two structures are identical up to the end of the NP "the students who missed the exam". There is no possible way to tell the two structures apart until the morphology of the verb can be examined.

While this and other such examples appear to be quite compelling, it is my belief that natural language need not be parsed by a mechanism that

simulates nondeterminism, that the seeming necessity for such an approach is an illusion. This paper is based upon this hypothesis; the central idea behind the research reported here is that *the syntax of any natural language can be parsed by a mechanism which operates "strictly deterministically" in that it does not simulate a nondeterministic machine*, in a sense which will be made precise below. (Actually, this paper will investigate this hypothesis only for the particular case of English; nothing will be said about languages other than English in what follows below.) I begin by assuming that this hypothesis is true, and then pursue the consequences of this assumption.

Two important restrictions on the range of this work should be made clear at the outset:

First, I intend to hypothesize only that the *syntactic component* operates strictly deterministically; as is discussed in [Marcus 77], there is a clear necessity for a strictly deterministic parser to ask questions of semantic/pragmatic components which by their very nature involve a limited amount of semantic parallelism. (Some of these semantic tests are *comparative* tests which involve the production of two competing semantic structures, one of which will ultimately be discarded. Because these tests cannot be nested, however, the process is not combinatoric.)

Second, this document will not attempt to *prove* that this hypothesis is true, i.e. I will not offer a proof that there are no constructions in English that violate this hypothesis. Instead, I will demonstrate that this assumption, which will be referred to as the Determinism Hypothesis, leads directly to a simple mechanism, a grammar interpreter, which has the following properties, among others:

-The grammar interpreter allows simple rules to be written which elegantly capture the significant generalizations behind such constructions as passives, *yes/no* questions, imperatives, and sentences with existential "there". These rules are reminiscent of the sorts of rules proposed within the framework of the theory of generative grammar, despite the fact that the rules presented here must recover underlying structure given only the terminal string of the surface form of the sentence. The component of the grammar interpreter which allows such rules to be formulated follows directly from the Determinism Hypothesis.

-The structure of the grammar interpreter constrains its operation in such a way that only very complex, *ad hoc* grammar rules can parse sentences which violate several of the constraints on rules of grammar proposed within the last several years by Chomsky. (The syntactic structures that the parser creates are by and large *Annotated Surface Structures* of the type currently proposed by Chomsky, complete with traces, although each parse node is also annotated with a set of features *a la* Winograd.) Most of the structural properties of the grammar interpreter upon which this result

depends are directly motivated by the Determinism Hypothesis.

-The grammar interpreter provides a simple explanation for the difficulty caused by "garden path" sentences, sentences like "The horse raced past the barn fell". In essence, the grammar interpreter allows special diagnostic rules to be written which can diagnose between the alternative cases presented in figure 1 above, but there is a limitation on the power of such rules which follows from a parameter of the mechanism. By appropriately setting this parameter, sentences like those in figure 1 can be diagnosed, but those which typically cause garden paths cannot. The component of the mechanism which this parameter affects is the same component that allows the formulation of the linguistic generalizations discussed above. (These garden path sentences clearly disconfirm one possible form of the Determinism Hypothesis which would say that all sentences which are *grammatical according to a purely competence grammar* can be parsed strictly deterministically, but the explanation for such sentences afforded by this model is consistent with a more "psychological" formulation of the hypothesis: that all sentences *which people can parse without conscious difficulty* can be parsed strictly deterministically.)

To the extent that these properties, all of which reflect deep properties of natural language, follow from the Determinism Hypothesis, and in this sense are explained by it, this paper provides indirect evidence for the truth of the hypothesis.

A Note on Methodology

From the properties of the grammar interpreter which this paper will investigate, it should be clear that the theory presented herein is an *explanatory* theory of language, rather than merely a *descriptive* theory. My central concern is not the particular grammar of English that has been developed in the course of this work, but rather the general properties of the grammar interpreter that enable a grammar of this form to be written. Thus, the central focus of what follows will not be on the particular rules of grammar presented as examples, but rather on the properties of the interpreter that allow rules like these to be written.

I will also be much concerned with what *must* be the case, given the Determinism Hypothesis as a starting assumption; I will attempt to show wherever possible that the mechanism I develop *necessarily* has the properties that it does, with the Determinism Hypothesis as the primary "forcing function". In this respect, I believe this research to be unique within the field of natural language processing.

Given this focus, this paper will not discuss details of the implementation of the parser presented here at all, and will only discuss particular rules of grammar when relevant to a higher level point. From the point of view of this paper, the implementation of the parser is only of interest as a "scratchpad" on which the model of the grammar interpreter and

the rules of grammar discussed here can be tested for adequacy.

This is not to say that the parser has not been implemented; indeed, *all of the snapshots of the parser's operation included in this document are taken directly from actual traces of the parser's operation and all of the code for grammar rules presented in this document has been tested in this implementation.* And again, this is not to say that the grammar interpreter does not have powerful engineering applications, but that these applications are not relevant to the theoretical issues considered here. Indeed, it should be noted that an early version of this parser and grammar have been used as a front end for the Personal Assistant Project at the MIT A.I. Lab [Goldstein & Roberts 77]. However, to reiterate the point, implementation issues are not the concern of this paper.

I believe that the justification of a theory of parsing must be in terms of its explanatory power (taking this term now in a wider sense than it is used in the theory of generative grammar) and the generalizations and universals of the competence linguists are one important form of explanation. However, there are clear differences between a theory of parsing and an abstract model of linguistic competence. On the one hand, any theory of parsing states how language is to be *processed*, going from the surface string to some underlying representation. On the other hand, the theory of generative grammar is formally a "proof checker" which states how to decide if some given string of tree structures (or, more precisely, phrase markers) is a legal derivation, in the formal-language theoretic sense of "derivation". As a processing model, there are criteria of adequacy for a parsing theory which go beyond - as well as potentially include - the criteria which are imposed upon the competence theories of generative grammar.

Thus, it is true that a parsing theory should attempt to capture wherever possible the sorts of generalizations that linguistic competence theories capture; there is no reason in principle why some of these generalizations should not be expressible in processing terms. On the other hand, a parsing theory can also be justified in terms of additional criteria including such issues as the adequacy of the theory as a psychological model, and the computational properties of the mechanism. These computational properties can include formal and informal limitations on resource utilization such as the time and space complexity of the computation, as exemplified by the Determinism Hypothesis. Other computational properties can include such issues as the net effect of interactions between various subsystems upon the overall computational complexity of the problem, viewed formally or informally.

The key point to be made, however, is that the search should be a search for universals, even - and perhaps especially - in the performance domain. For it would seem that the strongest parsing theory is one which says that *the grammar interpreter itself* is a universal mechanism, i.e. that there is one grammar interpreter which is the appropriate machine for

parsing *all* natural languages. If this is true, then properties of that machine should be reflected in the structure of each language, providing evidence that those properties are themselves reflections of the structure of the human mind.

The Notion of "Strictly Deterministic"

In the discussion above, the Determinism Hypothesis was loosely formulated as follows:

> Natural language can be parsed by a mechanism that operates "strictly deterministically" in that it does not simulate a nondeterministic machine....

The notion of "strictly deterministic" is central to the meaning of this hypothesis; exactly what does it mean?

To avoid any possible misunderstanding, let me state explicitly that the Determinism Hypothesis cannot mean simply that language can be parsed by a deterministic machine. As noted above, any computational mechanism that physically exists is deterministic in the automata theoretic sense, and thus any process which is specified by an algorithm for such a machine must be deterministic in this sense. From this it follows that any parser, whether it simulates a nondeterministic machine or not, must itself be deterministic. (The reader should note that "deterministic" does *not* mean non-probabilistic, nor does "nondeterministic" mean probabilistic. A nondeterministic machine, instead, can be conceptualized as a machine which has a magical oracle which tells it the right decision to make whenever its course of action is not strictly determined by the input and the state of the machine.)

Rather than attempting to formulate any rigorous, general explanation of what it means to "not simulate a nondeterministic machine", I will focus instead on three specific properties of the grammar interpreter which will be presented in this paper. These properties are special in that they will prevent this interpreter from simulating nondeterminism by blocking the implementation of either backtracking or pseudo-parallelism. This discussion is not intended to be definitional, but rather to give the reader a better grasp of the computational restrictions which will be embodied in the grammar interpreter, whose structure will be sketched in the next section. These three properties are:

1) *All syntactic substructures created by the machine are permanent.* This eliminates the possibility of simulating determinism by "backtracking", i.e by undoing the actions that were done while pursuing a guess that turns out to be incorrect.

2) *All syntactic substructures created by the machine for a given input must be output as part of the syntactic structure assigned to that input.* Since the structure the grammar interpreter will assign to a given input expresses exactly one syntactic analysis, this property eliminates the

possibility of simulating nondeterminism via pseudo-parallelism. If a machine were to simulate nondeterminism in this manner, it would necessarily create structures for the alternative analyses that would result from each of the paths such a machine pursues in parallel. (The reader may wonder how such an interpreter can handle true global ambiguity. The answer is that while such a machine can only build one syntactic analysis for a given input, and thus can only represent one interpretation of the input, it can also observe that the input was ambiguous, and flag the output analysis to indicate that this analysis is only one of a range of coherent analyses. Some external mechanism will then be needed to force the interpreter to reparse the input, taking a different analysis path, if the other consistent analyses are desired.)

3) *The internal state of the mechanism must be constrained in such a way that no temporary syntactic structures are encoded within the internal state of the machine.* While this does not mean that the machine's internal state must be limited to a finite state control (the grammar interpreter to be presented below uses a pushdown stack, among other control structures), it must be limited - at least in its use - in such a way that structure is not hidden in the state of the machine.

One immediate implication of all this is that a grammar for any interpreter which embodies these properties must constrain that interpreter from ever making a mistake, since the interpreter can only correctly analyze a given input if it never creates any incorrect structure. This means that such a grammar must at least implicitly specify how to decide what the grammar interpreter should do next, i.e. it can never leave the grammar interpreter with more than one alternative.

II. The Structure of the Grammar Interpreter

Motivation

Taking the Determinism Hypothesis as a given, an examination of natural language leads to a further set of properties which any deterministic grammar interpreter must embody. (*Henceforth, I will use the word "deterministic" to mean "strictly deterministic" in the sense discussed above. Please note this usage.*) In particular, any such interpreter *must* have the following properties:

1) It must be at least partially data driven, BUT....
2) It must be able to reflect expectations that follow from general grammatical properties of the partial structures built up during the parsing process.
3) It must have some sort of "look-ahead" facility, even if it is basically left-to-right.

To show that each of these properties is necessary, it suffices to show a pair of sentences of English that cannot be distinguished by a mechanism without the given property, but which speakers of English understand without difficulty. The sentences shown in figure 2 below provide crucial pairs for

each of the properties listed above.

The parser must:

Be partially data driven.

(1a) John went to the store.

(1b) How much is the doggie in the window?

Reflect expectations.

(2a) I called [$_{NP}$ John] [$_{S}$ to make Sue feel better].

(2b) I wanted [$_{S}$ John to make Sue feel better].

Have some sort of look-ahead.

(3a) Have [$_{S}$ the boys take the exam today].

(3b) Have [$_{NP}$ the boys] [$_{VP}$taken the exam today].

Figure 2 - Some examples which motivate the structure of the parser.

Almost by definition, a hypothesis driven parser cannot be deterministic, and thus a deterministic parser must necessarily be at least partially data driven. The essence of the problem is that any parser which is purely hypothesis driven, i.e. which is purely top-down, must hypothesize several nested levels of structure before positing any constituents which can be checked against the input string itself.

For example, a top-down parser, newly given an input, might begin by hypothesizing that the input is a sentence. It might then hypothesize that the input is a declarative, and therefore that the input begins with a noun phrase. Assuming that the input begins with a noun phrase, it might finally hypothesize that the NP begins with a determiner, a hypothesis which can be tested against the input string. By this point, the parser has created structures that correspond to the S and the NP, which will have to be discarded for at least some inputs. (These structures might be implicit in the state of the machine, but this is simply a matter of how the constituents are represented at this point in the parsing process.) To take a concrete example, even so different a pair of sentences as 2.1a and 2.1b above cannot both be deterministically analyzed by a hypothesis driven parser. The problem, of course, is simply that any hypothesis driven parser must either attempt to parse a given input as a declarative sentence, beginning, say, with an NP, before it attempts to parse it as a question, beginning with an auxiliary, or vice versa. Whatever order the parser imposes upon these two possibilities relative to each other, the clause type attempted first must be at least occasionally wrong. If a parser is to be deterministic, it must look before it leaps.

A deterministic parser cannot be entirely bottom-up, however. Any parser that is purely bottom-up must initially misparse one of the two sentences given as 2.2a and 2.2b above. The problem is that the string "John to make Sue feel better" can be analyzed in two different ways: as one constituent that is an infinitival complement, as in 2.2b, or as two unrelated constituents, as in 2.2a; in this latter case, the NP "John" is the

object of the verb and the phrase "to make Sue feel better" serves as an adverbial "purpose" clause. The difference in structure between 2.2a and 2.2b can be predicted, however, if the parser can note that "want" typically takes an infinitive complement, while "call" cannot take such a complement. Thus, a deterministic parser must have be capable of using whatever information and expectations can be gleaned from an examination of the structures that have been built up at any given point in the parsing process. If a parser is to operate deterministically, it must use such information to constrain the analysis imposed on the remainder of the input.

Finally, if a deterministic parser is to correctly analyze such pairs of sentences as 2.3a and 2.3b above, it cannot operate in an entirely left-to-right manner. As discussed earlier, it is impossible to correct assign structures to this pair of sentences before examining the morphology of the verb following the NP "the boys". These sentences can be differentiated, however, if the parser has a large enough "window" on the clause to see this verb; if the verb ends in "en" (in the simple case presented here), then the clause is a yes/no question, otherwise it is a imperative. Thus, if a parser is to be deterministic, it must have some constrained facility for look-ahead. *It must be stressed that unless this look-ahead ability is constrained in some manner, the determinism claim is vacuous.*

The Structure of the Parser - an Overview

This paper will present a grammar interpreter called *PARSIFAL*, whose structure is motivated by the three principles discussed above. This grammar interpreter maintains two major data structures: a pushdown stack of incomplete constituents called *the active node stack*, and a small three-place *constituent buffer* which contains constituents which are complete, but whose higher level grammatical function is as yet uncertain.

Figure 3 below shows a snapshot of the parser's data structures taken while parsing the sentence "John should have scheduled the meeting.". Note that the active node stack is shown growing *downward*, so that the structure of the stack reflects the structure of the emerging parse tree. At the bottom of the stack is an auxiliary node labelled with the features *modal*, *past*, etc., which has as a daughter the modal "should". Above the bottom of the stack is an S node with an NP as a daughter, dominating the word "John". There are two words in the buffer, the verb "have" in the first buffer cell and the word "scheduled" in the second. The two words "the meeting" have not yet come to the attention of the parser. (The structures of form "(PARSE-AUX CPOOL)" and the like will be explained below.)

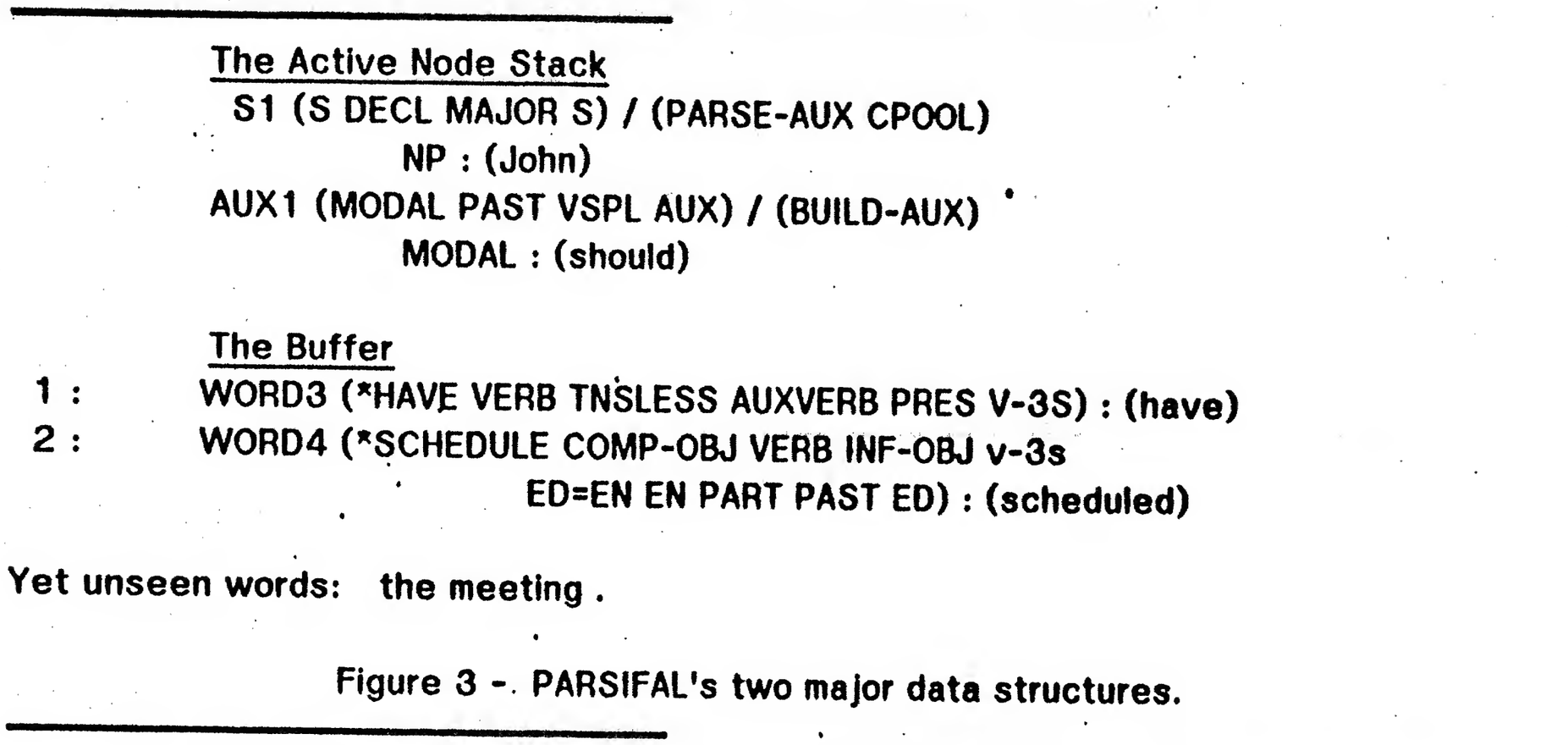

Figure 3 -. PARSIFAL's two major data structures.

The constituent buffer is the heart of the grammar interpreter; it is the central feature that distinguishes this parser from all others. The words that make up the parser's input first come to its attention when they appear at the end of this buffer after morphological analysis. Triggered by the words at the beginning of the buffer, the parser may decide to create a new grammatical constituent, create a new node at the bottom of the active node stack, and then begin to attach the constituents in the buffer to it. After this new constituent is completed, the parser will then pop the new constituent from the active node stack; if the grammatical role of this larger structure is as yet undetermined, the parser will insert it into the first cell of the buffer. The parser is free to examine the constituents in the buffer, to act upon them, and to otherwise use the buffer as a workspace.

In general, the parser uses the buffer in a first-in, first-out fashion. The availability of the buffer allows the parser to defer using the constituent in the leftmost buffer cell until it has a chance to examine one or two of the constituents to the immediate right of the first constituent. For example, the parser must often decide whether the word "have" at the beginning of a clause initiates a yes/no question, as in fig. 1.2b above, or an imperative, as in fig. 1.2a. The parser can often correctly decide what sort of clause it has encountered, and thus how to use the initial verb, by allowing several constituents to "pile up" in the buffer. Consider for example, the snapshot of the buffer shown in figure 4 below. By waiting until the NP "the boys", NP25, is formed, filling the 2nd buffer position, and WORD37, the verb "do", enters the buffer, filling the 3rd buffer position, the parser can see that the clause must be an imperative, and that "have" is therefore the main verb of the major clause.

WORD32	NP25	WORD37
HAVE	THE BOYS	DO

Figure 4 - The buffer allows the parser to examine local context.

While the buffer allows the parser to examine some of the context surrounding a given constituent, it does not allow arbitrary look-ahead. The length of the buffer is strictly limited; in the version of the parser presented here, the buffer has only three cells. (The buffer must be extended to five cells to allow the parser to build NPs in a manner which is transparent to the "clause level" grammar rules which will be presented in this paper. This extended parser still has a window of only three cells, but the effective start of the buffer can be changed through an "attention shifting mechanism" whenever the parser is building an NP. In effect, this extended parser has two "logical" buffers of length three, one for NPs and another for clauses, with these two buffers implemented by allowing an overlap in one larger buffer. For details, see [Marcus 77].)

Note that each of the three cells in the buffer can hold a *grammatical constituent* of any type, where a constituent is any tree that the parser has constructed under a single root node. The size of the structure underneath the node is immaterial; both "that" and "that the big green cookie monster's toe got stubbed" are perfectly good constituents once the parser has constructed a subordinate clause from the latter phrase.

The constituent buffer and the active node stack are acted upon by a grammar which is made up of pattern/action rules; this grammar can be viewed as an augmented form of Newell and Simon's production systems [Newell & Simon 72]. Each rule is made up of a pattern, which is matched against some subset of the constituents of the buffer and the accessible nodes in the active node stack (about which more will be said below), and an action, a sequence of operations which acts on these constituents. Each rule is assigned a numerical *priority*, which the grammar interpreter uses to arbitrate simultaneous matches.

The grammar as a whole is structured into *rule packets*, clumps of grammar rules which can be activated and deactivated as a group; the grammar interpreter only attempts to match rules in packets that have been activated by the grammar. Any grammar rule can activate a packet by associating that packet with the constituent at the bottom of the active node stack. As long as that node is at the bottom of the stack, the packets associated with it are active; when that node is pushed into the stack, the packets remain associated with it, but become active again only when that node reaches the bottom of the stack. For example, in figure 3 above, the

packet BUILD-AUX is associated with the bottom of the stack, and is thus active, while the packet PARSE-AUX is associated with the S node above the auxiliary.

The grammar rules themselves are written in a language called PIDGIN, an English-like formal language that is translated into LISP by a simple grammar translator based on the notion of top-down operator precedence [Pratt 73]. Figure 5 below gives a schematic overview of the organization of the grammar, and exhibits some of the rules that make up the packet PARSE-AUX.

Priority	Pattern Description of: 1st	2nd	3rd	The Stack	Action
		PACKET1			
5:	[ ]	[ ]	[ ]		--> ACTION1
10:	[ ]			[ ]	--> ACTION2
10:	[ ]	[ ]	[ ]	[ ]	--> ACTION3
		PACKET2			
10:	[ ]	[ ]			--> ACTION4
15:	[ ]			[ ]	--> ACTION5
		PACKET3			
5:	[ ]	[ ]			--> ACTION6
15:	[ ]	[ ]	[ ]		--> ACTION7

(a) - The structure of the grammar.

```
{RULE START-AUX PRIORITY: 10. IN PARSE-AUX
[=verb] -->
Create a new aux node.
Label C with the meet of the features of 1st and pres,
        past, future, tnsless.
Activate build-aux.}
{RULE TO-INFINITIVE PRIORITY: 10. IN PARSE-AUX
[=*to, auxverb] [=tnsless] -->
Label a new aux node inf.
Attach 1st to C as to.
Activate build-aux.}
{RULE AUX-ATTACH PRIORITY: 10. IN PARSE-AUX
[=aux] -->
Attach 1st to C as.aux.
Activate parse-vp. Deactivate parse-aux.}
```

(b) - Some sample grammar rules that initiate and attach auxiliaries.

Figure 5 - The structure of the grammar and some example rules.

The parser (i.e. the grammar interpreter interpreting some grammar) operates by attaching constituents which are in the buffer to the constituent at the bottom of the stack until that constituent is complete, at which time it is popped from the stack. If the constituents in the buffer provide sufficient evidence that a constituent of a given type should be initiated, a new node of that type can be created and pushed onto the stack; this new node can also be attached to the node at the bottom of the stack before the stack is pushed, if the grammatical function of the new constituent is clear when it is created. When popped, a constituent either remains attached to its parent, if it was attached to some larger constituent when it was created, or else it falls into the constituent buffer (which will cause an error if the buffer was already full).

This structure embodies the principles discussed above in the following ways:

1) *A deterministic parser must be at least partially data driven.* A grammar for PARSIFAL is made up of pattern/action rules which are triggered, in part, when lexical items or unattached larger constituents fulfilling specific descriptions appear in the buffer. Thus, the parser is directly responsive to the input it is given.
2) *A deterministic parser must be able to reflect expectations that follow from the partial structures built up during the parsing process.* Since PARSIFAL only attempts to match rules that are in active packets, grammar rules can activate and deactivate packets of rules to reflect the properties of the constituents in the active node stack. Thus grammar rules can easily be written that are constrained by whatever structure the parser is attempting to complete.
3) *A deterministic parser must have some sort of constrained look-ahead facility.* PARSIFAL's buffer provides this constrained look-ahead. Because the buffer can hold several constituents, a grammar rule can examine the context that follows the first constituent in the buffer before deciding what grammatical role it fills in a higher level structure. This document will argue that a buffer of quite limited length suffices to allow deterministic parsing. The key idea is that the size of the buffer can be sharply constrained if each location in the buffer can hold a single complete constituent, regardless of that constituent's size.

The Structure of Grammar Rules

We now turn to the structure of individual grammar rules. Some example rules were given in figure 5 above; this section will explain the syntax of those rules.

The pattern of a grammar rule is made up of a list of partial descriptions of parse nodes which must all be fulfilled if the pattern is to match. There can be up to five partial descriptions in each pattern: up to

three consecutive descriptions which are matched against the first, second, and third constituents in the buffer (in order), and two descriptions which match against the two nodes in the active node stack accessible to the parser. These two nodes are the bottom node on the stack, which will be referred to as the *current active node*, and the S or NP node closest to the bottom of the stack which will be called the *dominating cyclic node* or alternatively, if an S, the *current S node*. In figure 3 above, AUX1 is the current active node, and S1 is the current S node. As we shall see later, making the dominating cyclic node explicitly available for examination and modification seems to eliminate the need for any operations that ascend tree structures.

The syntax of the grammar rules presented in this paper should be self-explanatory for the most part, but a few comments on the grammar notation itself are in order. The general form of each grammar rule is:

{Rule <name> priority: <priority> in <packet>
<pattern> --> <action>}

Each pattern is of the form :

[<description of 1st buffer constituent>] [<2nd>] [<3rd>]

The symbol "=", used only in pattern descriptions, is to be read as "has the feature(s)". Features of the form "*<word>" mean "has the root <word>", e.g. "*have" means "has the root "have"". The tokens "1st", "2nd", "3rd" and "C" (or "c") refer to the constituents in the 1st, 2nd, and 3rd buffer positions and the current active node (i.e. the bottom of the stack), respectively. (I will also use these tags in the text below as names for their respective constituents.) The symbol "t" used in a pattern description is a predicate that is true of any node, thus "[t]" is the simplest always true description. Pattern descriptions to be matched against the current active node and the current S are flagged by "**C" appearing at the beginning of an additional pattern description. The PIDGIN code of the rule patterns should otherwise be fairly self-explanatory.

Each description is made up of a Boolean combination of tests for given grammatical features. Each description can also include Boolean feature tests on the daughters of the target node; the grammar language provides a tree walking notation for indicating specific daughters of a node. (The parser can access daughter nodes, but it cannot modify them.) While this richness of specification seems to be necessary, it should be noted that the majority of rules in even a moderately complex grammar have patterns which consist only of tests for the positive presence of given features.

The action of a grammar rule consists of a rudimentary program that does the actual work of building constituent structures. An action is built up of primitives that perform such actions as:

-*creating a new parse node*, pushing the newly created node onto the bottom of the active node stack. A new node is presumably created whenever the parser decides that the first constituent(s) in the buffer are the initial daughters of a constituent not yet created by the parser.

-*inserting a specific lexical item into a specific buffer cell*, which causes the previous contents of that cell and each cell to its right to be shifted one place to the right.

-*popping the current active node from the active node stack*, causing it to be inserted into the first buffer cell if it has not been previously attached to another node, shifting the previous contents of that cell one place to the right.

-*attaching a newly created node or a node in the buffer to the current active node or the current cyclic node*. After each grammar rule is executed, the grammar interpreter removes all newly attached nodes from the buffer, with nodes to the previous right of each deleted node shifting to the left. Note that a newly created node can either be attached to a parent node at the time of its creation, if its function in higher level grammatical structure is clear at that time, or it can be created without an attachment, in which case it will be dropped into the buffer when it is popped from the active node stack.

-*assigning features to a node in the buffer or one of the accessible nodes in the stack*.

-*activating and deactivating packets of rules*.

PIDGIN also provides primitives from which Boolean tests of the features of a node can be constructed, as mentioned above. These predicates can be used within "if...then...else..." expressions to conditionally perform various operations.

With the exception of allowing conditional expressions, PIDGIN rules fall into the simple class of programs called *fixed-instruction programs*. The PIDGIN language imposes the following constraints on rule actions:

1) There are no user-settable variables within the rule actions. The constituents in the first three buffer cells are available as the values of the parameters *1st*, *2nd* and *3rd* within each rule, but these parameters are given values by the grammar interpreter before each rule action is called, and are not resettable within an action. The values of the parameters C and *the current cyclic node* do change within a rule as nodes are pushed and popped from the active node stack, but their values cannot be set by a grammar rule.

2) PIDGIN allows no user-defined functions; the only functions within actions are PIDGIN primitives. (There is a limited ability for a rule to circumvent the pattern-matching process by explicitly naming its successor, but this is formally only a device for rule abbreviation, since the specification of the successor rule could simply be replaced with the code for the action of the rule named.)

3) While conditional "if...then....else..." expressions are allowed,

there is no recursion or iteration within actions.

4) The only structure building operations in PIDGIN are (a) attaching one node to another, and (b) adding features to a node's feature set. In particular, the list building primitives of LISP are *not* available in PIDGIN.

III. Parsing a Simple Declarative Sentence

In this section, I will present a small grammar which is just sufficient to parse a very simple declarative sentence, and then trace through the process of parsing the sentence (i) immediately below, given this grammar. The emphasis here will not be on the complexities of the grammar, but rather on the form of the grammar in broad outline and on the details of the workings of the grammar interpreter.

(i) John has scheduled a meeting.

One simplification will be imposed on this example: it will be assumed that all NPs come into the buffer pre-parsed, that the structure of NPs is determined by a mechanism that is transparent to the "clause-level" grammar rules that will be discussed here. Such a mechanism is in fact presented in [Marcus 77]; I will say here only that this mechanism involves relatively slight extensions of the mechanism presented in this paper.

As a convention, the parser begins every parse by calling the grammar rule named INITIAL-RULE. This rule creates an S node and activates the packet SS-START (Simple Sentence-START), which contains rules which decide on the type of simple sentences. The parser's state after this rule is executed is depicted in figure 6. At this point there is nothing in the buffer.

```
The Active Node Stack ( 0. deep)
C:      S16 (S) / (SS-START)

The Buffer

{RULE INITIAL-RULE IN NOWHERE
[t] -->
Create a new s node.
Activate ss-start.}
```

Figure 6 - After INITIAL-RULE has been run.

I remind the reader once again of the grammar interpreter's matching rules: that patterns are matched against the current contents of the buffer starting from its left edge, i.e. the first pattern description is matched against 1st, the second (if there is a second description in a given pattern) against 2nd, and the third (if there is one) against 3rd. The reader should also remember that constituents enter the buffer on demand; i.e. that the buffer mechanism will get the next constituent from the input word stream

when a rule pattern must be matched against a buffer cell that is currently empty. Furthermore, before the grammar interpreter will attempt to match a rule of a given priority, all rules with higher priority must explicitly fail to match. This means that throughout the course of the examples in this chapter, constituents will often enter the buffer for no apparent reason. These constituents were requested by rules that ultimately failed to match, leaving no trace of why each constituent entered the buffer. I will not comment further on the entry of constituents into the buffer.

The packet SS-START, some of whose rules are shown in figure 7 below, contains rules which determine the type of a major clause. If the clause begins with an NP followed by a verb, then the clause is labelled a declarative; if it begins with an auxiliary verb followed by an NP, it is labelled a yes/no question. If the clause begins with a tenseless verb, then not only is the clause labelled an imperative, but the word "you" is inserted into the buffer, where it is inserted at the beginning of the buffer by convention.

```
{RULE MAJOR-DECL-S IN SS-START
[=np] [=verb] -->
Label c s, decl, major.
Deactivate ss-start. Activate parse-subj.}

{RULE YES-NO-Q IN SS-START
[=auxverb] [=np] -->
Label c s, quest, ynquest, major.
Deactivate ss-start. Activate parse-subj.}

{RULE IMPERATIVE IN SS-START
[=tnsless] -->
Label c s, imper, major.
Insert the word 'you' into the buffer.
Deactivate ss-start. Activate parse-subj.}
```

Figure 7 - Some rules that determine sentence type.

After INITIAL-RULE has been executed and packet SS-START has been activated, the rule MAJOR-DECL-S matches, with the pattern matching process pulling the NP "John" and the verb "has" into the buffer. The action of the rule is now run, labelling the clause a major declarative clause, deactivating the packet SS-START, and activating the packet PARSE-SUBJ. The result of this is shown in figure 8 below.

The Active Node Stack (0. deep)
C: S16 (S DECL MAJOR S) / (PARSE-SUBJ)

The Buffer
1 : NP40 (NP NAME NS N3P) : (John)
2 : WORD125 (*HAVE VERB AUXVERB PRES V3S) : (has)

Yet unseen words: scheduled a meeting .

Figure 8 - After the rule MAJOR-DECL-S is run.

The packet PARSE-SUBJ contains rules which find and attach the subject of the clause under construction. It contains two major rules which are shown in figure 9 below. The rule UNMARKED-ORDER picks out the subject in clauses where the subject appears before the verb in surface order; the rule AUX-INVERSION picks out the subject in clauses where an element of the auxiliary occurs before the subject. Though the relevant rules will not be discussed here, the rule UNMARKED-ORDER will pick up the subject of imperatives and WH-questions where the subject of the clause is questioned, while AUX-INVERSION will pick up the subject of WH-questions that question other than the subject of the clause.

```
{RULE UNMARKED-ORDER IN PARSE-SUBJ
[=np] [=verb] -->
Attach 1st to c as np.
Deactivate parse-subj.
Activate parse-aux.}

{RULE AUX-INVERSION IN PARSE-SUBJ
[=auxverb] [=np] -->
Attach 2nd to c as np.
Deactivate parse-subj. Activate parse-aux.}
```

Figure 9 - Two subject-parsing rules.

The rule UNMARKED-ORDER now matches, and its action is run. This rule attaches 1st, i.e. NP40, the NP "John", to C, the node S16, as subject. It also activates the packet PARSE-AUX after deactivating PARSE-SUBJ. After this rule has been executed, the interpreter notices that the NP has been attached, and removes it from the buffer. Figure 10 shows the state of the parser after this rule is executed.

The Active Node Stack (0. deep)
C: S16 (S DECL MAJOR S) / (PARSE-AUX)
NP : (John)

The Buffer
1 : WORD125 (*HAVE VERB AUXVERB PRES V3S) : (has)

Yet unseen words: scheduled a meeting .

Figure 10 - After UNMARKED-ORDER has been executed.

For the sake of brevity the process of parsing the auxiliary verbs in this example will not be discussed, although figure 12 provides a trace of the application of rules during the parsing of the auxiliary phrase, through the attachment of the auxiliary to the VP node. The rules referred to in the trace are included in figure 11 below. The first frame of figure 12 results from rule START-AUX running with the parser in the state shown in figure 10 above. (The reader can either work his/her way through figure 12, or can safely continue reading after the trace.)

```
{RULE START-AUX PRIORITY: 10. IN PARSE-AUX .
        [=verb] -->
Create a new aux node.
Label C with the meet of the features of 1st and vspl, v1s,
        v+13s, vpl+2s, v-3s, v3s.
%(The above features are "person/number codes", e.g. "v1s"
flags a verb compatible with a 1st person subject.)%
Label C with the meet of the features of 1st and pres, past, future, tnsless.
Activate build-aux.}
{RULE AUX-ATTACH PRIORITY: 10. IN PARSE-AUX
        [=aux] --> Attach 1st to c as aux.
Activate parse-vp. Deactivate parse-aux.}
{RULE PERFECTIVE PRIORITY: 10. IN BUILD-AUX
        [=*have] [=en] --> Attach 1st to c as perf. Label c perf.}
{RULE PROGRESSIVE PRIORITY: 10. IN BUILD-AUX
        [=*be] [=ing] --> Attach 1st to c as prog. Label c prog.}
{RULE PASSIVE-AUX PRIORITY: 10. IN BUILD-AUX
        [=*be] [=en] --> Attach 1st to c as passive. Label c passive.}
{RULE AUX-COMPLETE PRIORITY: 15. IN BUILD-AUX
        [t] --> Drop c into the buffer.}
```

Figure 11 - Some rules which parse auxiliaries.

It should be noted that the rules of packet BUILD-AUX, some of which are shown in figure 11 above, are the equivalent of the transformational rule of affix-hopping. Note that these rules concisely state the relation between each auxiliary verb and its related affix by taking advantage of the ability to buffer both each auxiliary verb and the following

verb. It might seem that some patch to these rules is needed to handle question constructions in which, typically, the verb cluster is discontinuous, but this is not the case, as will be shown later in this paper.

```
About to run: PERFECTIVE
          The Active Node Stack ( 1. deep)
           S16 (S DECL MAJOR S) / (PARSE-AUX)
                    NP : (John)
C:         AUX14 (PRES V3S AUX) / (BUILD-AUX)

          The Buffer
1 :       WORD125 (*HAVE VERB AUXVERB PRES V3S) : (has)
2 :       WORD126 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...) : (scheduled)
Yet unseen words:  a meeting .

About to run: AUX-COMPLETE
          The Active Node Stack ( 1. deep)
           S16 (S DECL MAJOR S) / (PARSE-AUX)
                    NP : (John)
C:         AUX14 (PERF PRES V3S AUX) / (BUILD-AUX)
                    PERF : (has)

          The Buffer
1 :       WORD126 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...) : (scheduled)

About to run: AUX-ATTACH
          The Active Node Stack ( 0. deep)
C:         S16 (S DECL MAJOR S) / (PARSE-AUX)
                    NP : (John)

          The Buffer
1 :       AUX14 (PERF PRES V3S AUX) : (has)
2 :       WORD126 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...) : (scheduled)

          The Active Node Stack ( 0. deep)
C:         S16 (S DECL MAJOR S) / (PARSE-VP)
                    NP : (John)
                    AUX : (has)

          The Buffer
1 :       WORD126 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...) : (scheduled)
```

Figure 12 - Parsing the auxiliary of (i).

The packet PARSE-VP is now active. This packet contains, among other rules, the rule MVB (Main VerB), which creates and attaches a VP node and then attaches the main verb to it. This rule now matches and is run. The rule itself, and the resulting state of the parser, is shown in figure 13

below.

```
{RULE MVB IN PARSE-VP
[=verb] -->
Deactivate parse-vp.
If c is major then activate ss-final else
If c is sec then activate emb-s-final.
Attach a new vp node to c as vp.
Attach 1st to c %which is now the vp% as verb.
Activate subj-verb.}
```

```
                The Active Node Stack ( 1. deep)
                 S16 (S DECL MAJOR S) / (SS-FINAL)
                          NP : (John)
                          AUX : (has)
                          VP : ↓
C:               VP14 (VP) / (SUBJ-VERB)
                          VERB : (scheduled)

                The Buffer
1 :              WORD127 (*A NGSTART DET NS N3P ...) : (a)

Yet unseen words:  meeting .
```

Figure 13 - The rule MVB and the parser's state after its execution

At the time MVB is executed, the packet PARSE-VP is associated with S16, the current active node, as shown in the last frame of figure 12. The action of MVB first deactivates the packet PARSE-VP, then activates either SS-FINAL, if C is a major clause, or EMB-S-FINAL, if it is an embedded clause. These two packets both contain rules that parse clause-level prepositional phrases, adverbs, and the like. They differ in that the rules in EMB-S-FINAL must decide whether a given modifier should be attached to the current clause, or left in the buffer to be attached to a constituent higher up in the parse tree after the current active node is completed. Whatever packet is activated, the newly activated packet will be associated with C, S16, and thus the grammar interpreter will attempt to match the rules in it whenever S16 is the current active node.

The execution of the next line in the action of MVB results in a new VP node, VP14, being attached to S16 and then pushed onto the active node stack, becoming the current active node. Next the verb "scheduled", WORD126, is attached to the VP, and then the action of MVB activates the packet SUBJ-VERB. As is always the case with packet activation, this packet is associated with the node which is the current active node at the time of its activation, in this case VP14. Thus, this rule leaves the parser with the packet SS-FINAL associated with S16, and the packet SUBJ-VERB associated with VP14. Since VP14 is the current active node, SUBJ-VERB is now active.

Once VP14 is popped from the stack, leaving S16 as the current active node, SS-FINAL will be active.

The packet SUBJ-VERB contains rules which set up the proper environment for parsing the objects of the verb. The major rule in this packet, the rule SUBJ-VERB is shown in figure 14 below. Note that this rule has a very low priority (where 0 is the highest priority) and a pattern that will always match; this rule is a *default rule*, a rule which will become active when no other active rule can fire. While most of the code of the action of the rule SUBJ-VERB does not apply to our example, I have refrained from abbreviating this rule to give a feel for the power of PIDGIN.

```
{RULE SUBJ-VERB PRIORITY: 1'5. IN SUBJ-VERB
[t] -->
%Activate packets to parse objects and complements.%
If the verb of c is inf-obj then activate inf-comp.
If the verb of c is to-less-inf-obj then activate to-less-inf-comp.
If the verb of c is that-obj then activate that-comp.
If there is a wh-comp of the s above c
        and it is not utilized then activate wh-vp
        else if the s above c is major then activate ss-vp
        else activate embedded-s-vp.
Deactivate subj-verb.}
```

Figure 14 - The rule SUBJ-VERB.

The rule SUBJ-VERB is now the rule of highest priority that matches, so its action is now executed. Since the action of this rule is rather complex, I will discuss what it does in some detail.

The purpose of this rule is to activate the appropriate packets to parse the objects and complements of a clause; the activation of some of these packets depends upon the verb of the clause, while the activation of others depends upon more global properties of the clause, such as whether the clause is embedded or top-level. Thus, the next several lines of the action activate packets for various sorts of complement constructions that a verb might take: infinitive phrases in general (the packet INF-COMP), infinitive phrases that are not flagged by "to", e.g. "I saw John give Jane a kiss." (the packet TO-LESS-INF-COMP in addition to INF-COMP), and tensed complements (the packet THAT-COMP). The next long clause activates one of a number of packets which will attach the objects of the verb. The packet activated depends on the clause type: whether this clause still has a WH-head that needs to be utilized (WH-VP), whether this clause is secondary without a WH-head (EMBEDDED-S-VP), or whether this clause is a major clause (SS-VP).

The rule SUBJ-VERB provides a good example of one difference between the packets used to organize this grammar and the states of an ATN.

Packets do *not* simply correspond to ATN states, for several packets will typically be active at a time. For instance, if parsing the sentence "Who did John see Jane kiss?", this rule would activate three packets: INF-COMP, TO-LESS-INF-COMP, and WH-VP. In terms of the ATN model, one can think of this rule as dynamically tailoring the arcs out of a state of an ATN to exactly match various properties of the clause being parsed.

Of the complement-initiating packets, only the packet INF-COMP is activated in our example, since "schedule" can take infinitive complements, as in "Schedule John to give a lecture on Tuesday.". The packet SS-VP is then activated to attach the verb's objects, the packet SUBJ-VERB is deactivated, and the rule SUBJ-VERB is through. This rule thus changes the state of the parser only by activating and deactivating packets; the packets now associated with the current active node are SS-VP and INF-COMP. figure 15 below shows the rules in the packet SS-VP that will come into play in the example below. Note that the rule VP-DONE, like the rule SUBJ-VERB above, is a default rule, a rule which will become active when no other rule can apply, and will thus complete the VP when no other active rule can fire.

```
{RULE OBJECTS IN SS-VP
[=np] -->
Attach 1st to c as np.}

{RULE VP-DONE PRIORITY: 20 IN SS-VP
[t] --> Drop c.}
```

Figure 15

The rule OBJECTS is now triggered by NP41, and attaches the NP to VP14. The state of the parser at this point is shown in figure 16 below.

```
The Active Node Stack ( 1. deep)
S16 (S DECL MAJOR S) / (SS-FINAL)
                    NP : (John)
                    AUX : (has)
                    VP : ↓
C:      VP14 (VP) / (SS-VP INF-COMP)
                    VERB : (scheduled)
                    NP : (a meeting)

The Buffer
1 :     WORD133 (*. FINALPUNC PUNC) : (.)

Yet unseen words: (none)
```

Figure 16 - After the rule OBJECTS has been run

Completing the parse is now a simple matter. The default rule in packet SS-VP, VP-DONE now triggers, popping the VP node from the active node stack. Since VP14 is attached to S16, the S node above it on the stack, it remains hanging from the S node. The S node is once again the current active node, and packet SS-FINAL is now active. The default rule in this packet, SS-DONE, shown in figure 17, immediately triggers. This rule attaches the final punctuation mark to the clause, pops the S node from the stack, and signals the grammar interpreter that the parse is now complete, bringing our example to a close.

```
{RULE S-DONE  IN SS-FINAL
[=finalpunc] -->
Attach 1st to c as finalpunc.
%The next line is really part of the cf mechanism.%
Finalize the cf of s.
Parse is finished.}
```

Figure 17 - The rule S-DONE.

IV. CAPTURING LINGUISTIC GENERALIZATIONS

Introduction

In this section we will shift our attention from the grammar interpreter to the organization of the grammar itself and the form of the linguistic structures that it builds. This section will show that the grammar formalism is capable of capturing many of the same generalizations that are captured by traditional generative grammar, with much the same elegance.

The General Grammatical Framework - Traces

The form of the structures that the current grammar builds is based on the notion of *Annotated Surface Structure*. This term has been used in two different senses by Winograd [Winograd 71] and Chomsky [Chomsky 73]; the usage of the term here can be thought of as a synthesis of the two concepts. Following Winograd, this term will be used to refer to a notion of surface structure *annotated by the addition of a set of features* to each node in a parse tree. Following Chomsky, the term will be used to refer to a notion of surface structure annotated by the addition of an element called *trace* to indicate the "underlying position" of "shifted" NPs.

In current linguistic theory, a trace is essentially a "phonologically null" NP in the surface structure representation of a sentence that has no daughters but is "bound" to the NP that filled that position at some level of underlying structure. In a sense, a trace can be thought of as a "dummy" NP that serves as a placeholder for the NP that earlier filled that position; in the same sense, the trace's binding can be thought of as simply a pointer to that NP. In the context of this theory of parsing, I will define a trace simply to be an NP which has no daughters and which has associated with it a *binding register* which can be set to a pointer to another NP. (Within the grammar presented below, a trace will also be flagged by the feature *TRACE*.) This register can be set by the PIDGIN code

Set the binding of ⟨node⟩ to ⟨controlling node⟩.

It should be stressed at the outset that a trace is indistinguishable from a normal NP in terms of normal grammatical processes; a trace *is* an NP, even though it is an NP that dominates no lexical material.

Some examples of the use of trace are given in figure 18 immediately below.

(1a) What did John give to Sue?
(1b) What did John give *t* to Sue?
|_____|
(1c) John gave *what* to Sue.

(2a) A book was given Sue.
(2b) A book was given Sue *t*.
|_____|
(2c) ∇ gave Sue *a book*.

(3a) John was believed to be happy.
(3b) John was believed [$_S$ *t* to be happy].
|_____|

Figure 18 – Some examples of the use of trace.

One use of trace is to indicate the underlying position of the wh-head of a question or relative clause. Thus, the structure built by the parser for 18.1a would include the trace shown in 18.1b, with the trace's binding shown by the line under the sentence. The position of the trace indicates that 18.1a has an underlying structure analogous to the overt surface structure of 18.1c.

Another use of trace is to indicate the underlying position of the surface subject of a passivized clause. For example, 18.2a will be parsed into a structure that includes a trace as shown as 18.2b; this trace indicates that the subject of the passive has the underlying position shown in 18.2c. The symbol "∇" signifies the fact that the subject position of (2c) is filled by an NP that dominates no lexical structure. (Following Chomsky, I assume that a passive sentence in fact has *no underlying subject*, that an agentive "by NP" prepositional phrase originates as such in underlying structure.) The trace in (3b) indicates that the phrase "to be happy", which the brackets show is really an embedded clause, has an underlying subject which is identical with the surface subject of the matrix S, the clause that dominates the embedded complement. Note that what is conceptually the underlying subject of the embedded clause has been passivized into subject position of the matrix S, a phenomenon commonly called "raising". The analysis of this phenomenon assumed here derives from [Chomsky 73]; it is an alternative to the classic analysis which involves "raising" the subject of the embedded clause into object position of the matrix S before passivization (for details of this later analysis see [Postal 74]).

There are several reasons for choosing a properly annotated surface structure as a primary output representation for syntactic analysis. While a deeper analysis is needed to recover the predicate/argument structure of a sentence (either in terms of Fillmore case relations [Fillmore 68] or Gruber/Jackendoff "thematic relations" [Gruber 65; Jackendoff 72]),

phenomena such as focus, theme, pronominal reference, scope of quantification, and the like can be recovered only from the surface structure of a sentence. By means of proper annotation, it is possible to encode in the surface structure the "deep" syntactic information necessary to recover underlying predicate/argument relations, and thus to encode in the same formalism both deep syntactic relations and the surface order needed for pronominal reference and the other phenomena listed above.

Note that the information that these traces encode is easily used by a semantic component to recover the underlying predicate/argument structure of an utterance. For example, all that needs to be done to recover the predicate/argument structures of the (b) sentences in figure 18 above (where the predicate/argument structure is taken here loosely to be analogous to the structure of the (c) sentences) is to repeatedly replace each trace with its binding until all traces have been eliminated, and to ignore the surface subjects of passivized clauses.

While the notion of a trace and its binding is discussed above in terms of the theory of generative grammar, there is an essentially computational motivation for trace theory, as will be shown below. In essence, the point is this: Once a trace has been dropped into the buffer by a grammar rule, later grammatical processes will be unaware that an NP did not actually appear in this position in the input structure. Furthermore, by dropping a trace into the buffer, a grammar rule can assert, in effect, that an NP underlyingly appears at a given linear position in the input string without committing itself to the position of this NP in the underlying tree structure. This fact will turn out to be crucial to the formulation of the passive rule which will be presented below.

Some Captured Generalizations

In the remainder of this section, I will briefly sketch a few examples of grammar rules that explicitly capture, on nearly a one-to-one basis, the same generalizations that are typically captured by classical transformational rules. The central point I hope to make is that the availability of the buffer as a workspace, in conjunction with a grammar written in the form of pattern-action rules, makes possible several techniques for writing simple, concise grammar rules that have the net effect of explicitly "undoing" many of the generative grammarian's transformations with much the same elegance.

One *caveat* should be stated at the outset: Not all grammatical processes which are typically expressed as single rules within the generative framework can be so captured within the grammar for this parser, or, I believe, any other. Such processes include the general phenomenon of "WH-movement", which accounts for the structure of WH-questions and relative clauses (at the least), and the problem of prepositional phrase attachment. Thus, while there is a wide range of grammatical generalizations that can be captured within a parsing grammar, it must be conceded that there are

important generalizations that cannot be captured within this framework.

There are several techniques made possible by the buffer that will be used repeatedly to capture linguistic phenomena within fairly simple rule formulations. They are:

1) *The ability to remove some constituent other than the first from the constituent buffer*, compacting the buffer and reuniting discontinuous constituents. In natural language, it is often the case that some third structure intervenes between two parts of what is intuitively one constituent. In most parsers, special provisions must be made in the grammar for handling such situations. As will be demonstrated below, the buffer mechanism makes this unnecessary.

2) *The ability to place a trace by inserting it into the buffer* rather than by directly attaching it to a tree fragment. As I will sketch below, this yields a simple analysis of passivization and "raising".

3) *The ability to insert specific lexical items into the buffer*, thereby allowing one set of rules to operate on only superficially different cases. As figure 19 below shows, many grammatical constructions in natural language are best analyzed as slight variants of other constructions, differing only in the occurrence of an additional specific lexical item or two. Given the buffer mechanism, such constructions can be easily handled by doing a simple insertion of the appropriate lexical items into the shorter form of the construction, "transforming" the shorter form into the longer form, allowing both cases to then be handled by the same grammar rule.

1(a) all the boys
(b) all *of* the boys

2(a) I helped John pick it up.
(b) I helped John *to* pick it up.

Figure 19

In what follows below, I will give examples which illustrate points (1) and (2).

Example 1 - Yes/No Questions

In the grammar for this parser, the analysis of a yes-no question differs from the analysis of the related declarative only in the execution of two rules for each sentence type: declaratives trigger the two rules shown in figure 20a below, and yes-no questions trigger the two rules shown in 20b. The differences between the rules for declaratives and yes-no questions are underlined in fig. 20.

{RULE MAJOR-DECL-S IN SS-START
[=np] [=verb] -->
Label c s, decl, major.
Deactivate ss-start.
Activate parse-subj.}

{RULE UNMARKED-ORDER
IN PARSE-SUBJ
[=np] [=verb] -->
Attach 1st to c as np.
Deactivate parse-subj.
Activate parse-aux.}

DECLARATIVES
(a)

{RULE YES-NO-Q IN SS-START
[=auxverb] [=np] -->
Label c s, quest, ynquest, major.
Deactivate ss-start.
Activate parse-subj.}

{RULE AUX-INVERSION
IN PARSE-SUBJ
[=auxverb] [=np] -->
Attach 2nd to c as np.
Deactivate parse-subj.
Activate parse-aux.}

YES-NO QUESTIONS
(b)

Figure 20 - Grammar rules for yes-no questions and declaratives.

What is surprising about these rules is that they obviate the need for the grammar to contain special provisions to handle the discontinuity of the verb cluster in yes-no questions. Consider the following sentence,

(ii) Has John scheduled the meeting for Wednesday?

One of the auxiliary parsing rules that should be triggered during the course of the analysis of this sentence is the rule PERFECTIVE, shown below in figure 21. This rule will attach any form of "have" to the auxiliary and label the auxiliary with the feature *perfective* if the *following* word (implicitly a verb) has the feature *en*. It would seem that some provision must be made for the fact that in a yes-no question these two words might be separated by the subject NP, as in (ii) above where the verb "scheduled" (which does carry the feature *en*, since *en* is morphologically realized with this verb as "-ed") does not follow "has", but rather follows an intervening NP. As we shall see immediately below, however, no special patch is needed to handle this discontinuity at all.

{RULE PERFECTIVE PRIORITY: 10. IN BUILD-AUX
[=*have] [=en] --> Attach 1st to c as perf. Label c perf.}

Figure 21 - The PERFECTIVE rule requires contiguous verbs.

Let us now trace through the initial steps of parsing (ii) and see why it is that no changes to the auxiliary parsing rules are required to parse yes-no questions.

We begin with the parser in the state shown in figure 22a below, with the packet SS-START active. The rule YES-NO-Q matches and is

executed, labelling S17 with features that indicate that it is a yes-no question, as shown in figure 22b below. (The buffer, not shown again, remains unchanged.) This step of the parsing process is quite analogous to the analysis process for declaratives.

The Active Node Stack (0. deep)
C: S17 (S) / (SS-START)

The Buffer
1 : WORD134 (*HAVE VERB AUXVERB PRES V3S) : (Has)
2 : NP43 (NP NAME NS N3P) : (John)
3 : WORD136 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...) : (scheduled)

Yet unseen words: a meeting for Wednesday ?

(a) - Before YES-NO-Q has been executed.

The Active Node Stack (0. deep)
C: S17 (S QUEST YNQUEST MAJOR) / (PARSE-SUBJ)

(b) - The Active Node Stack after YES-NO-Q is executed.

Figure 22

The packet PARSE-SUBJ is now active, and rule AUX-INVERSION matches and is executed; it attaches NP43 to S17. After AUX-INVERSION has been executed, the grammar interpreter notices that NP43 is attached and *it therefore removes NP43 from the buffer.* But now that the subject of the clause has been removed from the buffer, the pieces of the verb cluster "has scheduled" are no longer discontinuous, as the word "has" is now in the 1st buffer cell, and "scheduled" is in the 2nd cell. This is shown in figure 23 below. In effect, the rule AUX-INVERSION, merely by picking out the subject of the clause, has "undone" the subject/auxiliary "inversion" which signals the presence of a question.

The Active Node Stack (0. deep)
C: S17 (S QUEST YNQUEST MAJOR) / (PARSE-AUX)
NP : (John)

The Buffer
1 : WORD134 (*HAVE VERB AUXVERB PRES V3S) : (Has)
2 : WORD136 (*SCHEDULE COMP-OBJ VERB INF-OBJ V-3S ...) : (scheduled)

Yet unseen words: a meeting for Wednesday ?

Figure 23 - After AUX-INVERSION has been executed.

From this simple example, we see that the ability to attach constituents in other than the first place in the buffer to the current active node, in conjunction with the fact that attachment causes a node to be removed from the buffer, compacting the remaining contents of the buffer, allows a key generalization to be captured. Most interestingly, the removal of the subject NP in this case was *not* specifically stipulated by the grammar rule, which merely specified that the second NP in the buffer was to be attached to the dominating S. The deletion followed instead from *general principles of the grammar interpreter's operation.* This latter point is crucial; given a simple statement of the structure of yes-no questions in English, the proper behavior follows from much more general principles.

Example 2 - Passives and Raising

In this section, I will very briefly sketch a grammatical solution to the phenomena of passivization and "raising" [Postal 74], sentences in which what seems to be the subject of an embedded complement is passivized into the subject position of the higher clause. This analysis, I believe, is simpler than that demonstrated by Woods within his classic paper on the ATN formalism [Woods 70], in that (1) nothing like the register mechanism of the ATN and the related SENDR and LIFTR mechanisms are needed for this solution; and (2) the register resetting involved in Woods' solution is not needed here.

Let us begin with the parser in the state shown in figure 24 below, in the midst of parsing the following sentence:

The meeting was scheduled for Wednesday.

The analysis process for the sentence prior to this point is essentially parallel to the analysis of any simple declarative with one exception: the rule PASSIVE-AUX in packet BUILD-AUX (shown in figure 11) has decoded the passive morphology in the auxiliary and given the auxiliary the feature *passive* (although this feature is not visible in figure 24). At the point we begin our example, the packet SUBJ-VERB is active.

```
                The Active Node Stack ( 1. deep)
                  S21 (S DECL MAJOR) / (SS-FINAL)
                            NP : (The meeting)
                            AUX : (has been)
                            VP : ↓
C:              VP17 (VP) / (SUBJ-VERB)
                            VERB : (scheduled)

                The Buffer
1 :             PP14 (PP) : (for Wednesday)
2 :             WORD162 (*. FINALPUNC PUNC) : (.)
```

Figure 24 - Analysis of a passive after the verb has been attached.

The packet SUBJ-VERB contains, among other rules, the rule PASSIVE, shown in figure 25 below. As I will show in the next section, this rule by itself is sufficient to account for many of the phenomena that accompany clause-level passivization including the phenomenon of raising. The pattern of this rule is fulfilled if the auxiliary of the S node dominating the current active node (which will always be a VP node if packet SUBJ-VERB is active) has the feature *passive*, and the S node has not yet been labelled *np-preposed*. (The notation "** C" indicates that this rule matches against the two accessible nodes in the stack, not against the contents of the buffer.) The action of the rule PASSIVE simply creates a trace, sets the binding of the trace to the subject of the dominating S node, and then drops the new trace into the buffer.

```
{RULE PASSIVE IN SUBJ-VERB
[** c; the aux of the s above c is passive;
        the s above c is not np-preposed] -->
Label the s above c np-preposed.
Create a new np node labelled trace.
Set the binding of c to the np of the s above c.
Drop c.}
```

Figure 25 - PASSIVE captures np-preposing in 4 lines of code.

The state of the parser after this rule has been executed, with the parser previously in the state in figure 24 above, is shown in figure 26 below. S21 is now labelled with the feature *np-preposed*, and there is a trace, NP53, in the first buffer position. NP53, as a trace, has no daughters, but is bound to the subject of S21.

The Active Node Stack (1. deep)
S21 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (has been)
VP : ↓
C: VP17 (VP) / (SUBJ-VERB)
VERB : (scheduled)

The Buffer
1 : NP53 (NP TRACE) : bound to: (The meeting)
2 : PP14 (PP) : (for Wednesday)
3 : WORD162 (*. FINALPUNC PUNC) : (.)

Figure 26 - After PASSIVE has been executed.

Now rules will run which will activate the two packets SS-VP and INF-COMP, given that the verb of VP17 is "schedule". These two packets contain rules for parsing simple objects of non-embedded Ss, and infinitive complements, respectively. Two such rules, each of which utilize an NP immediately following a verb, are given in figure 27 below. The rule OBJECTS, in packet SS-VP, picks up an NP after the verb and attaches it to the VP node as a simple object. The rule INF-S-START1, in packet INF-COMP, triggers when an NP is followed by "to" and a tenseless verb; it initiates an infinitive complement and attaches the NP as its subject. (An example of such a sentence is "We scheduled John to give a seminar next week".) The rule INF-S-START1 must have a higher priority than OBJECTS because the pattern of OBJECTS is fulfilled by any situation that fulfills the pattern of INF-S-START1; if both rules are in active packets and match, the higher priority of INF-S-START1 will cause it to be run instead of OBJECTS.

```
{RULE OBJECTS PRIORITY: 10 IN SS-VP
[=np] -->
Attach 1st to c as np.}

{RULE INF-S-START1 PRIORITY: 5. IN INF-COMP
[=np] [=*to,auxverb] [=tnsless] -->
Label a new s node sec, inf-s.
Attach 1st to c as np.
Activate parse-aux.}
```

Figure 27 - Two rules which utilize an NP following a verb.

While there is not space to continue the example here in detail, note that the rule OBJECTS will trigger with the parser in the state shown in figure 26 above, and will attach NP53 as the object of the verb "schedule. OBJECTS is thus totally indifferent both to the fact that NP53 was not a regular NP, but rather a trace, and the fact that NP53 did not originate in

the input string, but was placed into the buffer by grammatical processes. Whether or not this rule is executed is absolutely unaffected by differences between an active sentence and its passive form; the analysis process for either is identical as of this point in the parsing process. Thus, the analysis process will be exactly parallel in both cases after the PASSIVE rule has been executed. (I remind the reader that the analysis of passive assumed above, following Chomsky, does *not* assume a process of "agent deletion", "subject postposing" or the like.)

Example 3 - Passives in Embedded Complements

In the previous section, we investigated how simple passives were handled by the parser. In this section we will investigate how the parser handles much more complicated passives, such as the sentence presented below as 28a. The desired analysis of this sentence, as shown in 28b, analyzes "to be happy" as an embedded S whose subject is a trace bound to the subject of the higher clause. Such an analysis views 28b as deriving conceptually from an underlying form similar to 28c, where the subject of the underlying clause has been passivized into subject position in the upper clause. In this section I will show how the parser builds such an analysis.

(a) John was believed to be happy.
(b) John was believed [$_S$ *t* to be happy].
|_____|
(c) ∇ believed *John* to be happy.

Figure 28 - The intended analysis for a complex passive .

The reader may have wondered why the PASSIVE rule introduced in the previous section was formulated to drop the trace it creates into the buffer rather than immediately attaching the new trace to the VP node. As I will show in this section, such a formulation of PASSIVE also correctly analyzes passives like 28c above which involve "raising", but with no additional complexity added to the grammar, correctly capturing a generalization about a range of grammatical phenomena. To fully show the range of the generalization, the example which we will investigate in this section, sentence (1) in figure 29 below, is yet a level more complex than 28a above; its analysis is shown schematically in 29.2. In this example there are two traces: the first, the subject of the embedded clause, is bound to the subject of the major clause, the second, the object of the embedded S, is bound to the first trace, and is thus ultimately bound to the subject of the higher S as well. Thus the underlying position of the NP "the meeting" can be viewed as being the object position of the embedded S, as shown in 29.3.

(1) The meeting was believed to have been scheduled for Wednesday.
(2) The meeting was believed [$_S$ *t* to have been scheduled t for Wednesday]
|_____||_____|
(3) ∇ believed [$_S$ ∇ to have scheduled *the meeting* for Wednesday].

Figure 29 - The example which will be discussed in this section.

We begin our example, once again, right after the rule MVB has been executed, attaching "believed" to VP20, the current active node, as shown in figure 30 below. Note that "was" has been attached to the AUX node, and that the AUX node has been labelled *passive*, although this feature is not shown here.

The Active Node Stack (1. deep)
S22 (S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (was)
VP : ↓
C: VP20 (VP) / (SUBJ-VERB)
VERB : (believed)

The Buffer
1 : WORD166 (*TO PREP AUXVERB) : (to)
2 : WORD167 (*HAVE VERB TNSLESS AUXVERB PRES ...) : (have)

Figure 30 - The example begins after MVB has been executed.

With the parser in the state indicated in figure 30 above, the packet SUBJ-VERB, which contains PASSIVE, is active, and this rule's pattern is fulfilled, so the rule is executed. This rule, as stated above, creates a trace, binds it to the subject of the current clause, and drops the trace into the first cell in the buffer. The resulting state is shown in figure 31 below.

The Active Node Stack (1. deep)
S22 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (was)
VP : ↓
C: VP20 (VP) / (SUBJ-VERB)
VERB : (believed)

The Buffer
1 : NP55 (NP TRACE) : bound to: (The meeting)
2 : WORD166 (*TO PREP AUXVERB) : (to)
3 : WORD167 (*HAVE VERB TNSLESS AUXVERB PRES ...) : (have)

Yet unseen words: been scheduled for Wednesday .

Figure 31 - After PASSIVE has been executed.

The rule SUBJ-VERB is now triggered, and deactivates the packet SUBJ-VERB and activates the packet SS-VP (which contains the rule OBJECTS) and, since "believe" takes infinitive complements, the packet INF-COMP (which contains INF-S-START1), among others. Now the patterns of OBJECTS and INF-S-START1 will both match, and INF-S-START1, shown above in figure 27, will be executed by the interpreter since it has the higher priority. (Note once again that a trace is a perfectly normal NP from the point view of the pattern matching process.) This rule now creates a new S node labelled infinitive and attaches the trace NP55 to the new infinitive as its subject. The resulting state is shown in figure 32 below.

The Active Node Stack (2. deep)
S22 (NP-PREPOSED S DECL MAJOR) / (SS-FINAL)
NP : (The meeting)
AUX : (was)
VP : ↓
VP20 (VP) / (SS-VP THAT-COMP INF-COMP)
VERB : (believed)
C: S23 (SEC INF-S S) / (PARSE-AUX)
NP : bound to: (The meeting)

The Buffer
1 : WORD166 (*TO PREP AUXVERB) : (to)
2 : WORD167 (*HAVE VERB TNSLESS AUXVERB PRES ...) : (have)

Yet unseen words: been scheduled for Wednesday .

Figure 32 - After INF-S-START1 has been executed.

We are now well on our way to the desired analysis. An embedded infinitive has been initiated, and a trace bound to the subject of the dominating S has been attached as its subject.

The parser will now proceed, exactly as in earlier examples, to build the auxiliary, attach it, and attach the verb "scheduled" to a new VP node. After the rules that accomplish this have been executed, the parser is left in the state depicted in figure 33 below. (Note that for the sake of brevity, only the 3 bottommost nodes in the active node stack will be shown in this and all successive diagrams.) The infinitive auxiliary has been parsed and attached, and VP21 is now the current active node, with the verb "scheduled" as main verb of the clause. It should be noted that the auxiliary has been assigned the feature *passive* by the auxiliary parsing rules, although this is not shown in the figure below.

The Active Node Stack (3. deep)

.....

VP20 (VP) / (SS-VP THAT-COMP INF-COMP)
VERB : (believed)
S23 (SEC INF-S S) / (EMB-S-FINAL)
NP : bound to: (The meeting)
AUX : (to have been)
VP : ↓
C: VP21 (VP) / (SUBJ-VERB)
VERB : (scheduled)

The Buffer

1 : PP15 (PP) : (for Wednesday)
2 : WORD174 (*. FINALPUNC PUNC) : (.)

Figure 33 - After the auxiliary and main verb have been parsed.

The packet SUBJ-VERB, containing the rules PASSIVE and SUBJ-VERB, is now active. Once again PASSIVE's pattern matches and this rule is executed, creating a trace, binding it to the subject of the clause, (which is in this case itself a trace), and dropping the new trace into the buffer. And again, it labels the dominating S node, in this case S23, with the feature *np-preposed*, blocking the PASSIVE rule from reapplying. This is shown in figure 34 below. Note that in this figure, as in earlier figures, the lexical NP which is the transitive closure of the binding relationship is shown for each trace.

The Active Node Stack (3. deep)

.....

VP20 (VP) / (SS-VP THAT-COMP INF-COMP)
VERB : (believed)
S23 (NP-PREPOSED SEC INF-S S) / (EMB-S-FINAL)
NP : bound to: (The meeting)
AUX : (to have been)
VP : ↓
C: VP21 (VP) / (SUBJ-VERB)
VERB : (scheduled)

The Buffer

1 : NP57 (NP TRACE) : bound to: (The meeting)
2 : PP15 (PP) : (for Wednesday)
3 : WORD174 (*. FINALPUNC PUNC) : (.)

Figure 34 - After PASSIVE has run on the lower clause.

The remainder of the parsing process proceeds in a fashion similar to the simple passive example discussed above, with the rule OBJECTS next attaching the trace NP57 as the object of VP21.

Once the infinitive complement is completed, it is dropped into the buffer, where it triggers a grammar rule which attachs it to an NP node attached to the VP of the major clause. The tree structure which results after the parse is complete is shown in figure 35 below. (For the sake of brevity, most features have been deleted from this tree.) A trace is indicated in this tree by giving the terminal string of its ultimate binding in parentheses.

```
(NP-PREPOSED S DECL MAJOR)
     NP: (MODIBLE NP DEF DET NP)
          DET: The
          NBAR: (NS NBAR)
               NOUN: meeting
     AUX: (PAST V13S AUX)
          PASSIVE: was
     VP: (VP)
          VERB: believed
          NP: (NP COMP)
               S: (NP-PREPOSED SEC INF-S S)
                    NP: (NP TRACE)
                         (bound* to: The meeting)
                AUX: (PASSIVE PERF INF AUX)
                     TO: to
                     PERF: have
                     PASSIVE: been
                VP: (VP)
                     VERB: scheduled
                     NP: (NP TRACE)
                              (bound* to: The meeting)
                     PP: (PP)
                          PREP: for
                          NP: (NP TIME DOW)
                               NOUN: Wednesday
FINALPUNC: .
```

Figure 35 - The tree structure resulting from parsing 29.1.

We have seen that the simple formulation of the PASSIVE rule presented above, interacting with other simply formulated grammatical rules for parsing objects and initiating embedded infinitives, allows a trace to be attached either as the object of a verb or as the subject of an embedded infinitive, whichever is the appropriate analysis for a given grammatical situation. The PASSIVE rule is formulated in such a way that it drops the trace it creates into the buffer, rather than attaching the trace somewhere in particular in the tree. Because of this, later rules, already formulated to trigger on an NP in the buffer, will analyze sentences with NP-preposing exactly the same as those without a preposed subject. The PASSIVE rule, formulated as it is, elegantly captures the notion that the "underlying" location of a preposed NP is, in fact, immediately after the verb in the underlying *terminal string*, regardless of its position in the underlying *tree*. Once again, we see that the availability of the buffer mechanism is crucial to capturing this generalization; such a generalization can only be stated by a parser with a mechanism much like the buffer used here.

V. The Grammar Interpreter and Chomsky's Constraints

Introduction

In a series of papers over the last several years, Noam Chomsky has argued for several specific properties of language which he claims are universal to all human languages [Chomsky 73, 75, 76]. These properties, which form one of the cornerstones of his current linguistic theory, are embodied in a set of constraints on language, a set of restrictions on the operation of rules of grammar.

In this section, I will give some examples that demonstrate that important sub-cases of two of these constraints fall out naturally from the structure of the grammar interpreter. I will then show that many of the properties of the interpreter which are crucial to capturing these universals are motivated by the Determinism Hypothesis. Thus, I will demonstrate that significant sub-cases of Chomsky's purported universals follow naturally from the Determinism Hypothesis and in this sense are explained by it. Such a demonstration provides further evidence in favor of the Determinism Hypothesis.

(It should be noted that these constraints are far from universally accepted. They are currently the source of much controversy; for various critiques of Chomsky's position see [Postal 74; Bresnan 76]. I will not argue for these constraints here but will accept them as a given.)

Before preceding with the argument itself, let me briefly sketch the two constraints, Subjacency and the Specified Subject Constraint, that will be the focus of this section.

Subjacency

Before defining the notion of Subjacency, a few auxiliary technical terms need to be defined: If we can trace a path up the tree from a given node X to a given node Y, then we say *X is dominated by Y*, or equivalently, *Y dominates X*. If Y dominates X, and no other nodes intervene (i.e. X is a daughter of Y), then *Y immediately (or directly) dominates X*. [Akmajian & Heny 75]. One non-standard definition will prove useful: I will say that if Y dominates X, and Y is a cyclic node, i.e. an S or NP node, and there is no other cyclic node Z such that Y dominates Z and Z dominates X (i.e. there is no intervening cyclic node Z between Y and X) then *Y Dominates X*.

The principle of Subjacency, informally stated, says that no rule can involve constituents that are separated by more than one cyclic node. Let us say that a node X *is subjacent to* a node Y if there is at most one cyclic node, i.e. at most one NP or S node, between the cyclic node that Dominates Y and the node X. Given this definition, the Subjacency principle says that no rule can involve constituents that are not subjacent.

The Subjacency principle implies that Chomsky's two movement rules, the rules MOVE NP and MOVE WH-phrase, are constrained so that they can move a constituent only into positions that the constituent was subjacent to. This means that if α, β, and ϵ in figure 36 are cyclic nodes, no rule can move a constituent from position X to either of the positions Y, where $[_\alpha ... X ...]$ is distinct from $[_\alpha X]$.

$[_\epsilon ... Y ... [_\beta ... [_\alpha ... X ...] ...] ... Y ...]$

Figure 36 - Subjacency: no rule can involve X and Y in this structure.

Subjacency implies that if a constituent is to be "lifted" up more than one level in constituent structure, this operation must be done by repeated operations. Thus, to use one of Chomsky's examples, the sentence given in figure 37a, with a deep structure analogous to 37b, must be derived as follows (assuming that "is certain", like "seems", has no subject in underlying structure): The deep structure must first undergo a movement operation that results in a structure analogous to 37c, and then another movement operation that results in 37d, each of these movements leaving a trace as shown. That 37c is in fact an intermediate structure is supported by the existence of sentences such as 37e, which purportedly result when the ∇ in the matrix S is replaced by the lexical item "it", and the embedded S is tensed rather than infinitival. The structure given in 37f is ruled out as a possible annotated surface structure, because the single trace could only be left if the NP "John" was moved in one fell swoop from its underlying position to its position in surface structure, which would violate Subjacency.

(a) John seems to be certain to win.
(b) ∇ seems $[_S \nabla$ to be certain $[_S$ John to win]]
(c) ∇ seems $[_S$ John to be certain $[_S$ *t* to win]]
(d) John seems $[_S$ *t* to be certain $[_S$ *t* to win]]
(e) It seems that John is certain to win.
(f) John seems $[_S \nabla$ to be certain $[_S$ *t* to win]]

Figure 37 - An example demonstrating Subjacency.

The Specified Subject Constraint

The Specified Subject Constraint (SSC), stated informally, says that no rule may involve two constituents that are Dominated by different cyclic nodes unless the lower of the two is the subject of its S or NP. Thus, no rule may involve constituents X and Y in the structure shown in figure 38 below, if α and β are cyclic nodes and Z is the subject of α, Z distinct from X.

$[_{\beta}...Y...[_{\alpha}\ Z...X...]...Y...]$

Figure 38 - SSC: No rule can involve X and Y in this structure.

The SSC explains why the surface subject position of verbs like "seems" and "is certain" which have no underlying subject can be filled only by the subject and not the object of the embedded S: The rule MOVE NP is free to shift any NP into the empty subject position, but is constrained by the SSC so that the object of the embedded S cannot be moved out of that clause. This explains why (a) in figure 39 below, but not 39b, can be derived from 39c; the derivation of 39c would violate the SSC.

(a) John seems to like Mary.
(b)*Mary seems John to like.
(c) ∇ seems $[_S$ John to like Mary]

Figure 39 - Some examples illustrating the SSC.

Limitations

Before turning to the aspects of these constraints that are accounted for by the operation of the grammar interpreter, there are several important limitations of this work which must be discussed.

First of all, the range of phenomena for which explanations can be given in terms of the structure of the grammar interpreter is more limited than that accounted for by Chomsky's constraints. While two of Chomsky's constraints, the Specified Subject Constraint and Subjacency, seem to fall out of the grammar interpreter, there seems to be no apparent account of a third, the Tensed S (or Propositional Island) Constraint, in terms of this mechanism.

Second, I show here only that movement rules are constrained by the grammar interpreter. Chomsky's constraints themselves are intended to apply to all rules of grammar, both syntactic rules (i.e. transformations) and those rules of semantic interpretation which Chomsky now calls "rules of construal", a set of shallow semantic rules which govern anaphoric processes [Chomsky 77]. The discussion here will only touch on purely syntactic phenomena; the question of how rules of semantic interpretation can be meshed with the framework presented in this document is beyond the range of the present research. This would seem to be a fertile area for future investigation. For more on all of this, the non-linguist is encouraged to read Chapter 3 of [Chomsky 75], which is an introductory presentation of the notions outlined here.

Third, the arguments presented below deal only with English, and in fact depend strongly upon several facts about English syntax (e.g. the fact that English is subject-initial). Whether these arguments can be successfully

extended to other languages is an open question, and to this extent this work must be considered exploratory.

And finally, I will not show that these constraints must be true *without exception*; as we will see, there are various situations in which the constraints imposed by the grammar interpreter can be circumvented. Most of these situations, though, will be shown to demand much more complex grammar formulations than those typically needed in the grammar so far constructed. This is quite in keeping with the suggestion made by Chomsky [Chomsky 77] that the constraints are not necessarily without exception, but rather that exceptions will be "highly marked" and therefore will count heavily against any grammar that includes them.

Because of space limitations, this section deals only with those grammatical processes characterized by the competence rule "MOVE NP"; the constraints imposed by the grammar interpreter upon those processes characterized by the rule "MOVE WH-phrase" are discussed at length in [Marcus 77]. There I show that the behavior characterized by Ross's Complex NP Constraint [Ross 67] itself follows directly from the structure of the grammar interpreter for rather different reasons than the behavior considered in this section. But this is not the problem it might seem: As Chomsky himself notes, the phenomena he characterizes by the rule "MOVE WH-phrase" do not, in fact, appear to observe his constraints. Instead, Chomsky argues that his constraints account for the behavior stipulated by the Complex NP Constraint. The claim then, is that the grammar interpreter accounts for the same range of behavior (given the caveats above) that Chomsky's two constraints account for, but in rather different ways for MOVE NP and MOVE Wh-phrase phenomena.

Also because of space limitations, I will not attempt to show that the two constraints I will deal with here *necessarily* follow from the grammar interpreter, but rather only that they *naturally* follow from the interpreter. In particular, I will show that they follow from the simple, natural formulation of PASSIVE presented in the previous section, whose formulation itself depends heavily upon the structure of the interpreter. Again, necessity is argued for in detail in [Marcus 77].

The Specified Subject Constraint and the Grammar Interpreter

As I pointed out above, the Specified Subject Constraint constrains the rule "MOVE NP" in such a way that only the subject of a clause can be moved out of that clause into a position in a higher S. Thus, if a trace in an annotated surface structure is bound to an NP Dominated by a higher S, that trace must fill the subject position of the lower clause. In this section I will show that the grammar interpreter constrains grammatical processes in such a way that annotated surface structures constructed by the grammar interpreter will have this same property.

In terms of the parsing process, this means that if a trace is "lowered" from one clause to another during the parsing process, then it will be attached as the subject of the second clause, unless it is created by a WH-movement process. To be more precise, if a trace is attached so that it is Dominated by S1, and the trace is bound to an NP Dominated by some other S node S2, then that trace will necessarily be attached so that it fills the subject position of S1. This is depicted in figure 40 below. This will be true presupposing the structural analyses for passivization and other constructions involving traces presented in previous chapters and assuming, again, that the grammar does not violate a small set of restrictions.

The Active Node Stack

.....

S2 ... / ...

...

NP2

...

C: S1 ... / ...

NP: NP1 (NP TRACE) : bound to NP2

Figure 40 - NP1 must be attached as the subject of S1 since it is bound to an NP Dominated by a higher S.

Let us briefly review the last example in section IV, which showed how the trace *t1* in (iii) below was created, bound to the NP "the meeting", and attached as the subject of the embedded clause.

(iii) The meeting was believed [$_S$ *t1* to have been scheduled for Wednesday]

Immediately after the verb "believed" was attached to the VP of the major clause, the PASSIVE rule was executed. This rule created a trace, bound it to the subject of the dominant S, and then dropped this new trace into the first position in the buffer. If this were a simple passive, the OBJECT rule would now attach the trace to the VP of the current clause. In this case, however, the rule INF-S-START1, with pattern

[=np] [=*to] [=tnsless]

triggered, creating a new S node and then attaching the trace to that S node as its subject. The essence of the process is as follows: create and properly bind a trace while the major S is the current S; drop the trace into the buffer; create a subordinate S; attach the trace to the newly created S.

The end result of this process is that a trace bound to an NP in a higher S has been attached as the subject of an embedded S without having been explicitly "lowered" from one S to the other. The original point of the example, of course, was that the rather simple PASSIVE rule handles both

this case and the case of simple passives without the need for some mechanism to explicitly lower the NP. The PASSIVE rule captures this generalization by dropping the trace it creates into the buffer (after appropriately binding the trace), thus allowing other rules written to handle normal NPs to correctly place the trace.

This statement of PASSIVE does more, however, than simply capture a generalization about a specific construction. As I will argue in detail below, the behavior specified by both the Specified Subject Constraint and Subjacency follows almost immediately from this formulation. In [Marcus 77], I argue that this formulation of PASSIVE is the only simple, non-*ad hoc*, formulation of this rule possible, and that all other rules characterized by the competence rule "MOVE NP" must operate similarly; in that paper I attempt to show that all other possible formulations are blocked by the nature of the grammar interpreter. However, while it is easy to show that the behavior characterized by the SSC follows from the PASSIVE rule as stated here, the demonstration that other possible formulations of PASSIVE are blocked by the structure of the grammar interpreter is quite tedious. Because of this, I will only show here that these constraints follow naturally from this formulation of PASSIVE, leaving the question of necessity aside. Here I will assume one additional constraint, the *Left-to-Right Constraint* and then show that the SSC and Subjacency naturally follow from this constraint, given the Passive rule stated above. I will then briefly motivate this constraint as a natural condition on the formulation of a grammar for this mechanism.

> The Left-to-Right Constraint: the constituents in the buffer are (almost always) attached to higher level constituents in left-to-right order, i.e. the first constituent in the buffer is (almost always) attached before the second constituent.

I will now show that a trace created by PASSIVE which is bound to an NP in one clause can only serve as the subject of a clause dominated by that first clause.

Given the formulation of PASSIVE, a trace can be "lowered" into one clause from another only by the indirect route of dropping it into the buffer before the subordinate clause node is created, which is exactly how the PASSIVE rule operates. This means that the ordering of the operations is crucially: 1) create a trace and drop it into the buffer, 2) create a subordinate S node, 3) attach the trace to the newly created S node. The key point is that at the time that the subordinate clause node is created and becomes the current active node, the trace must be sitting in the buffer, filling one of the three buffer positions. Thus, the parser will be in the state shown in figure 41 below, with the trace, in fact, most likely in the first buffer position.

The Active Node Stack

.....

C: S123 (S SEC ...) / ...

The Buffer

...

NP123 (NP TRACE) : bound to NP in S above S123

...

Figure 41 - Parser state after embedded S created.

Now, given the L-to-R Constraint, a trace which is in the buffer at the time that an embedded S node is first created must be one of the first several constituents attached to the S node or its daughter nodes. From the structure of English, we know that the leftmost three constituents of an embedded S node, ignoring topicalized constituents, must either be

COMP NP AUX
or
NP AUX [$_{VP}$ VERB ...].

(The COMP node will dominate flags like "that" or "for" that mark the beginning of a complement clause.) But then, if a trace, itself an NP, is one of the first several constituents attached to an embedded clause, the only position it can fill will be the subject of the clause.

This is all that must be demonstrated to capture the behavior characterized by the SSC. I have shown that any trace bound to an NP in a higher clause which is attached to a lower clause can only be attached as the subject of that clause. This is exactly the empirical consequence of Chomsky's Specified Subject Constraint in such cases as explained above.

The L-to-R Constraint

I will now briefly discuss the motivation for the L-to-R Constraint. I will not attempt to prove that this constraint must be true, but merely to show why it is plausible.

For the grammar of English discussed in this paper, and, it would seem, for any grammar of English that attempts to capture the same range of generalizations as this grammar, the constituents in the buffer are attached to the current active node (and thus are removed from the buffer by the grammar interpreter) in left-to-right order, with a small range of exceptions. This usage is clearly not enforced by the grammar interpreter as presently implemented; it is quite possible to write a set of grammar rules that specifically ignores a constituent in the buffer until some arbitrary point in the clause, though such a set of rules would be highly *ad hoc*. However, there rarely seems to be a need to remove other than the first constituent in

the buffer.

For this reason, I will make the following assumption in the discussion that follows: the constituents in the buffer are almost always attached to higher level constituents in left-to-right order, i.e. the first constituent in the buffer is almost always attached before the second constituent. The one exception to this seems to be that a constituent C_i may be attached before the constituent to its left, C_{i-1}, if C_i does not appear in surface structure in its underlying position (or, if one prefers, in its unmarked position) and if its removal from the buffer reestablishes the unmarked order of the remaining constituents, as in the case of the AUX-INVERSION rule discussed earlier in this paper. To capture this notion, the L-to-R Constraint can be restated as follows: All constituents must be attached to higher level constituents according to the left-to-right order of constituents in the unmarked case of that constituent's structure. This restatement seems, empirically, to include most of the exceptions I can think of to the constraint as stated above.

This reformulation is interesting in that it would be a natural consequence of the operation of the grammar interpreter if packets were associated with the phrase structure rules of the base, and these rules were used as templates to build up the structure assigned by the grammar interpreter, thereby adding a "base component" of phrase structure rules to the grammar. A packet of grammar rules would then be explicitly associated with each symbol on the right hand side of each phrase structure rule. A constituent of a given type would then be constructed by activating the packets associated with each node type of the appropriate phrase structure rule in left-to-right order. Since these base rules would reflect the unmarked l-to-r order of constituents, the constraint suggested here would then simply fall out of the interpreter mechanism.

Subjacency and the Grammar Interpreter

Chomsky's Subjacency constraint, stated informally, says that no rule can affect constituents in two distinct clauses (or more generally, two cyclic constituents) S1 and S2 unless S1 Dominates S2 (i.e. S1 dominates S2 and there is no third clause S3 that "comes between" S1 and S2). In terms of the competence rule "MOVE-NP", this means that an NP can be moved only within the clause in which it originates, or into the clause that Dominates that clause. In this section I will show that the parsing correlate of this constraint follows from the structure of the grammar interpreter. Specifically, I will show that there are only limited cases in which a trace generated by a "MOVE-NP" process can be "lowered" more than one clause, i.e. that a trace created and bound while any given S is current must almost always be attached either to that S or to an S which is Dominated by that S.

Let us begin by examining what it would mean to lower a trace more than one clause. Given that a trace can only be "lowered" by dropping it into the buffer and then creating a subordinate S node, as discussed above,

lowering a trace more than one clause necessarily implies the following sequence of events, depicted in figure 42 below: First, a trace NP1 must be created with some S node, S1, as the current S, bound to some NP Dominated by that S and then dropped into the buffer. By definition, it will be inserted into the first cell in the buffer. (This is shown in figure 42a) Then a second S, S2, must be created, supplanting S1 as the current S, and then yet a third S, S3, must be created, becoming the current S (fig. 42b). During all these steps, the trace NP1 remains sitting in the buffer. Finally, NP1 is attached under S3 (fig. 42c). By the Specified Subject Constraint, NP1 must then attach to S3 as its subject.

The Active Node Stack

.....

C: S1 ... / ...

The Buffer

1st: NP1 (NP TRACE) : bound to NP Dominated by S1

...

(a) - NP1 is dropped into the buffer while S1 is the current S.

The Active Node Stack

.....

S1 ... / ...

S2 ... / ...

C: S3 ... / ...

The Buffer

...

NP1 (NP TRACE) : bound to NP Dominated by S1

...

(b) - S2 and then S3 are created, with NP1 still in the buffer.

The Active Node Stack

.....

S1 ... / ...

S2 ... / ...

C: S3 ... / ...

NP: NP1 (NP TRACE) : bound to NP Dominated by S1

The Buffer

...

(c) - NP1 is attached to S3 as its subject (by the SSC).

Figure 42 - Lowering a trace more than 1 clause

But this sequence of events is highly unlikely. The essence of the argument is this:

Nothing in the buffer can change between the time that S2 is created and S3 is created if NP1 remains in the buffer. NP1, like any other node that is dropped from the active node stack into the buffer, is inserted into the first buffer position. But then, by the L-to-R Constraint, nothing to the right of NP1 can be attached to a higher level constituent until NP1 is attached. (One can show that it is most unlikely that any constituents will enter to the left of NP1 after it is dropped into the buffer, but I will suppress this detail here; the full argument is included in [Marcus 77].)

But if the contents of the buffer do not change between the creation of S2 and S3, then what can possibly motivate the creation of both S2 and S3? The contents of the buffer must necessarily provide clear evidence that both of these clauses are present, since, by the Determinism Hypothesis, the parser must be correct if it initiates a constituent. Thus, the same three constituents in the buffer must provide convincing evidence not only for the creation of S2 but also for S3. Furthermore, if NP1 is to become the subject of S3, and if S2 Dominates S3, then it would seem that the constituents that follow NP1 in the buffer must also be constituents of S3, since S3 must be completed before it is dropped from the active node stack and constituents can then be attached to S2. But then S2 must be created *entirely* on the basis of evidence provided by the constituents of another clause (unless S3 has less than three constituents). Thus, it would seem that the contents of the buffer cannot provide evidence for the presence of both clauses unless the presence of S3, by itself, is enough to provide confirming evidence for the presence of S2. This would be the case only if there were, say, a clausal construction that could only appear (perhaps in a particular environment) as the initial constituent of a higher clause. In this case, if there are such constructions, a violation of Subjacency should be possible.

With the one exception just mentioned, there is no motivation for creating two clauses in such a situation, and thus the initiation of only one such clause can be motivated. But if only one clause is initiated before NP1 is attached, then NP1 must be attached to this clause, and this clause is necessarily subjacent to the clause which Dominates the NP to which it is bound. Thus, the grammar interpreter enforces the Subjacency Constraint.

As a concluding point, it is worthy of note that while the grammar interpreter appears to behave exactly as if it were constrained by the Subjacency principle, it is in fact constrained by a version of the Clausemate Constraint! (The Clausemate Constraint, long tacitly assumed by linguists but first explicitly stated, I believe, by Postal [Postal 64], states that a transformation can only involve constituents that are Dominated by the same cyclic node. This constraint is at the heart of Postal's attack on the constraints that are discussed above and his argument for a "raising" analysis.) The grammar interpreter, as was repeatedly stated above, limits

grammar rules from examining any node in the active node stack higher than the current cyclic node, which is to say that it can only examine clausemates. This parsing version of the Clausemate Constraint, in fact, is crucial to the argument presented above showing that Subjacency is a natural consequence of the grammar interpreter. The trick is that a trace is created and bound while it is a "clausemate" of the NP to which it is bound in that the current cyclic node at that time is the node to which that NP is attached. The trace is then dropped into the buffer and another S node is created, thereby destroying the clausemate relationship. The trace is then attached to this new S node. Thus, in a sense, the trace *is* lowered from one clause to another. The crucial point is that while this lowering goes on as a result of the operation of the grammar interpreter, it is only implicitly lowered in that 1) the trace was never *attached* to the higher S and 2) it is *not* dropped into the buffer because of any realization that it must be "lowered"; in fact it may end up attached as a clausemate of the NP to which it is bound - as the passive examples presented earlier make clear. The trace is simply dropped into the buffer because its grammatical function is not clear, and the creation of the second S follows from other independently motivated grammatical processes. From the point of view of this performance theory, we can thus have our cake and eat it too; to the extent that it makes sense to map results from the realm of performance into the realm of competence, in a sense *both* the clausemate/"raising" and the Subjacency positions are correct.

These Arguments as Evidence in Support of the Determinism Hypothesis

In closing, I would like to show that the properties of the grammar interpreter crucial to capturing the behavior of Chomsky's constraints were originally motivated by the Determinism Hypothesis, and thus, to some extent, the Determinism Hypothesis explains Chomsky's constraints.

The strongest form of such an argument, of course, would be to show that (a) either (i) the grammar interpreter accounts for *all* of Chomsky's constraints in a manner which is conclusively universal or (ii) the constraints that it will not account for are wrong and that (b) the properties of the grammar interpreter which were crucial for this proof were *forced* by the Determinism Hypothesis. (To show necessity, of course, it must be demonstrated either that no other mechanism would serve to implement a deterministic parser or that any such mechanism would lead to fundamentally the same results.) If such an argument could be made, it would show that those of Chomsky's constraints which are valid follow from the Determinism Hypothesis, giving strong confirmation to the Determinism Hypothesis.

I have shown none of the above, and thus my claims must be proportionately more modest. I have argued only that important sub-cases of Chomsky's constraints follow from the grammar interpreter, and while I can show that the Determinism Hypothesis strongly *motivates* the mechanisms

from which these arguments follow, I cannot show necessity. The extent to which this argument provides evidence for the Determinism Hypothesis must thus be left to the reader; no objective measure exists for such matters.

The ability to drop a trace into the buffer is at the heart of the arguments presented above for Subjacency and the SSC as consequences of the functioning of the grammar interpreter; this is the central operation upon which the above arguments are based. Also crucial to the arguments are various restrictions upon the operation of the grammar interpreter, such as the fact that there is no way to access any nodes in the active node stack except for the current active node and the current cyclic node.

As discussed earlier in this paper, the buffer itself, and the fact that a constituent can be dropped into the buffer if its grammatical function is uncertain, are directly motivated by the Determinism Hypothesis. It was argued above that a parser must necessarily provide some sort of look-ahead mechanism if it is to function deterministically. It was also argued that this look-ahead must be constrained in some manner if the determinism claim was to have any content, but that this constraint must be based on some number of constituents, rather than some fixed number of lexical items. It is easy to see that such a requirement also necessarily implies an ability to construct constituents and - in some sense or other - buffer them; the example given in that earlier discussion demonstrated the necessity of holding on to the first three constituents of a clause before the grammatical role of the first could be determined. Given these attributes which a deterministic parser *must* fulfill, the buffer mechanism intuitively seems to be one of the simplest computational devices that fulfills these specifications. In this sense, it seems clear that the buffer mechanism and the ability to drop constituents from the active node stack into the buffer follow naturally and directly from the general principles which a deterministic parser must fulfill.

It is thus the case that the aspects of the grammar interpreter which are crucial to the arguments presented above, namely the buffer mechanism and its ability to buffer a constituent until its grammatical function is clear, are directly motivated by the Determinism Hypothesis. Given this, it is fair to claim that if Chomsky's constraints follow from the operation of the grammar interpreter, then they are strongly linked to the Determinism Hypothesis. If Chomsky's constraints are in fact true, then the arguments presented in this chapter provide solid evidence in support of the Determinism Hypothesis.

VI. Differential Diagnosis and Garden Path Sentences

Introduction

As the observant reader has probably noted, there are sentences in English which seem to indicate that *people* don't parse English deterministically, quite independently of any claim that English *can* be parsed

that way. While such sentences are not *a priori* counterexamples to the Determinism Hypothesis as stated in part 1 of this paper, they seem to call into question the validity of the Determinism Hypothesis as a basis for any sort of psychological modelling. In this section I will present some anecdotal evidence which supports speculation that such sentences are in fact the exception that proves the rule, given one small modification of the Determinism Hypothesis:

> There is enough information in the structure of natural language in general, and in English in particular, to allow left-to-right deterministic parsing of those sentences *which a native speaker can analyze without conscious effort.*

This section will present a potential explanation for the difficulty caused by so-called *"garden path" sentences*, sentences which have perfectly acceptable syntactic structures, yet which many readers initially attempt to analyze as some other sort of construction, i.e. sentences which lead the reader "down the garden path". (Some examples of garden path (GP) sentences are given in figure 43 below.) This theory will derive from an investigation of several situations in which a deterministic parser is presented with an input which contains a *local structural ambiguity*. (An ambiguity is *local* if it can be resolved by examining the grammatical context provided by the entire sentence; an ambiguity is *structural* if two different structures can be built up out of a string of smaller constituents each of a given type.) To handle each of these ambiguities, a special purpose grammar rule will be presented which can *differentially diagnose* between the two structural possibilities. I will show that some ambiguities can be diagnosed with assurance by PARSIFAL's grammar rules while it seems that diagnosing other ambiguities requires a buffer "window" larger than the three constituent maximum which each rule is allowed to examine. These latter ambiguities, I suggest, are exactly those which cause garden paths.

(a) The grocery store always orders a hundred pound bags of sugar.
(b) I told the boy the dog bit Sue would help him.
(c) The horse raced past the barn fell.

Figure 43 - Some examples of garden path sentences.

It must be stressed again that the theory which will be presented here is highly speculative, and that the evidence presented for it is either anecdotal or the result of informal experiment. Hopefully, such "armchair psycho-linguistics" will provide incentive for future research in this area which will yield more substantial evidence for or against the speculations presented below.

Diagnosing Between Imperatives and Yes/No Questions

I will now turn to the question of what causes a sentence to cause a parser to take a garden path. This will be done in the context of a

specific structural ambiguity and the diagnostic rule which attempts to resolve it. Investigation of where and why this rule fails will lead directly to a theory of garden paths.

Consider the following sentences, first presented early in this paper:

(iii)a Have the students who missed the exam take the exam today.
(iii)b Have the students who missed the exam taken the exam today?

Even though (iii)a is an imperative sentence, with "have" as the main verb of the matrix S, and (iii)b is a yes/no question, with "have" as an auxilliary, the initial segments of these two sentences are exactly identical. Because these two constituents serve in different grammatical roles in the two different constructions, neither of these two constituents can be utilized by a deterministic parser until it is determined whether the sentence is an imperative or a yes/no question.

Within the framework of the PARSIFAL grammar interpreter, the initial segment of these sentences is ambiguous in that it fulfills the patterns of both IMPERATIVE and YES/NO-QUESTION, both of which are in packet SS-START with identical priorities. (These rules were shown previously in figure 7.) This is because the word "have" (as well as the word "can") can be both an auxiliary verb, fulfilling the first node description in the pattern of YES/NO-QUESTION, or a tenseless verb, fulfilling the first node description in the pattern of IMPERATIVE. To resolve this ambiguity, either an additional diagnostic rule must be added to the grammar or else one of these two rules must be changed to eliminate the ambiguity altogether. In some sense, the two choices are equivalent, but the first is conceptually the cleaner of the two solutions, and it is this option which will be developed in the discussion that follows.

To begin with, it is clear that this diagnostic rule must apply whenever both IMPERATIVE and YES/NO-QUESTION are applicable, i.e. exactly when the word "have" appears in the first buffer position and an NP appears in the second position, as discussed above. Furthermore, the diagnostic must have access to the constituent in the third buffer location, since the pair of sentences given in (iii) above differ only in this third position. Given this, the pattern of the diagnostic rule, which I will call HAVE-DIAGNOSTIC, must be:

[=*have, tnsless] [=np] [t]

(The form of the word "have" that is tenseless is the word "have", so the feature *auxverb* need not be specified in this pattern.)

Second, it is clear that a sentence which fulfills this pattern can be a yes/no question only if the NP, which will be the subject of the sentence, agrees with the initial verb in person and number. Since the verb "have"

will agree with an NP of any person and number except 3rd person singular, if the NP *is* 3rd person singular, the sentence must be an imperative. For example, in figure 44 below, any clause whose initial segment is as shown in (a) must be an imperative, while any clause whose initial segment is as shown in 44.b may be either an imperative or a yes/no-question. (This is the first mention of person/number agreement in this document. Usually, it is unnecessary to use agreement information to decide upon the proper interpretation of a sentence, and so agreement can be safely ignored. Where it seems to be of value is in resolving various ambiguous situations such as this. I believe that agreement information will become highly important when an attempt is made to resolve lexical ambiguity deterministically, but that issue is outside the scope of this document.)

(a) Have the student who missed the exam .....
(b) Have the students who missed the exam ....

Figure 44 - Sentence (a) must be an imperative.

If the NP is not 3rd person singular, then HAVE-DIAGNOSTIC must examine the constituent in the 3rd buffer position and try to determine the correct analysis for the input sentence on the basis of its features. If 3rd is an NP, then the sentence is a yes/no-question, as in .Figure 45.a below. If 3rd is *tnsless* (i.e. tenseless), then the sentence is an imperative, as in 45.b.

(a) Have the boys *a dollar between them*?
(b) Have the students who missed the exam *take* it today.

Figure 45 - Diagnosis on the basis of the 3rd constituent.

One might suppose that if 3rd is *en*, as in (iii)b above, that the sentence must be a yes/no-question, but this is not the case, as the following sentence shows:

(iv) Have the students who booed taken to the principal!

If this feature occurs, the sentence is either a yes/no-question, or else an imperative with a passive subordinate clause, but one cannot tell which without examining more of the sentence. Indeed, it is possible for such a sentence to be globally ambiguous (ignoring final punctuation), as the sentences (v)a and (v)b show:

(v)a Have all of the eggs broken?
(v)b Have all of the eggs broken!

In terms of the grammar interpreter, it is clear that more than three constituents must be examined to diagnose such pairs of sentences (using final punctuation to resolve the above pair). This seems to call into question the constraint which allows a grammar rule to examine only the first three

constituents in the grammar interpreter's buffer.

For reasons which will become clear below, I will ignore this newly revealed problem, and simply stipulate, for the moment, that if HAVE-DIAGNOSTIC cannot determine what the structure of a given input sentence is on the basis of the three constituents that it can examine, then it will guess that the sentence is a yes/no-question. Assuming that the diagnostic tests discussed above reveal as much information as can be gleaned from an examination of only the first three buffer constituents, the code for the rule HAVE-DIAGNOSTIC can now be given; this diagnostic is shown in Figure 9.10 below. Since this rule must be run instead of either YES/NO-QUESTION or IMPERATIVE if all three rules are applicable, it must be in the same packet as they, and it must have higher priority than either of them.

```
{RULE HAVE-DIAG PRIORITY: 5 IN SS-START
[=*have, tnsless] [=np] [t] -->
If 2nd is ns, n3p or 3rd is tnsless
        then run imperative next else
If 3rd is not verb then run yes-no-q next else
%If not sure, assume it's a y/n-q and%
        run yes-no-q next.}
```

Figure 9.10 - HAVE-DIAGNOSTIC summarizes the diagnostics discussed above.

Garden Paths and a Three Constituent Window

Let us now turn to the stipulation that each grammar rule can only examine the first three constituents in the buffer.

In the next few sections, I will present some evidence which supports the contention that this stipulation is empirically justified. This evidence results from a series of experiments conducted by the author during the course of this research. As these experiments were highly informal in nature, the conclusions presented below can be taken as highly suggestive at best, although the results were typically quite clear.

In each of these experiments, some number of people were asked to read a sentence or a series of sentences typed on a sheet of paper. After reading each sentence, each person was asked if he encountered any difficulty while reading the sentence, or, if he was familiar with the notion, if the sentence was a garden path. At least 15 people were asked to look at each of these sentences, unless the first four or five people who were asked to examine a particular sentence either all or none took a garden path on that sentence. Somewhere over twenty such experiments were conducted, at frequent intervals, over a three year period, as I discovered more and more possible garden paths while developing grammar rules. About two thirds of the sentences people were asked to read led to garden paths for at least half the people asked. Approximately 50 or 60 people total participated in at least

one of these experiments; most were students or staff at the MIT Artificial Intelligence Laboratory.

Let me now claim, on the basis of several of these experiments, that the diagnostic rule HAVE-DIAGNOSTIC resolves the relevant ambiguity approximately as well as many people, i.e. that many people, given a sentence which would trigger this diagnostic if input to the parser, will arbitrarily analyze it as a yes/no-question if they cannot diagnose its structure on the basis of the first three constituents. If the sentence is actually an imperative, these people will eventually realize that they were led down the garden path, and will consciously reanalyze the sentence.

In the most surprising of the relevant experiments, approximately 40 people were asked to read the following sentence, and then asked if they had any difficulty the first time they read the sentence, i.e. whether they noticed taking a garden path on the sentence:

(vi) Have the packages delivered tomorrow.

Of the 40 people asked, 20 reported misanalyzing the sentence as a question at least for an instant, and more than a few were quite startled when they first realized that this simple sentence did not have the structure they had supposed. (It should be noted, by the way, that the sentence is entirely anomalous if interpreted as a yes/no question, yet still fully half the people shown this sentence misanalyzed it initially. This provides some evidence that high level semantic analysis is *not* employed to diagnose such ambiguities. Several people, when asked about this, mentioned imagining a package giving birth to little packages in an attempt to semantically reconcile the false yes/no-question interpretation!)

This result immediately leads one to ask why half the people shown sentence (vi) *do* analyze the sentence correctly. One possible explanation is that people simply choose one of the two possibilities at random when they are faced with such an undiagnosable ambiguity. If this is true, those people who did not notice taking a garden path while analyzing the sentence (vi) simply happened to guess the correct structure initially, which seems quite plausible.

Unfortunately, this simple explanation seems to be wrong, as the following experiment shows. About fifteen people were asked to read the sentence (vii) below, and see if they had any difficulty in understanding what it meant:

(vii) Have the soldiers given their medals by their sweethearts.

Only two of the fifteen had no difficulty with this sentence, and four or five had to be shown the correct analysis. (The sentence can be paraphrased "Have their sweethearts give the soldiers their medals.".) Almost without

exception, everyone shown the sentence tried to interpret it as a yes/no question. This result is consistent with other experiments involving this ambiguity as well; people almost never seem to mistakenly interpret a yes/no question as an imperative. Thus, almost without exception, if a person cannot resolve this ambiguity, he assumes the yes/no-question interpretation. This result disproves the simple "random-guess" theory suggested above.

One explanation consistent with both the above experiments seems to be the following, although I have no evidence to offer in its favor: The number of buffer cells that can be examined by a grammar rule, and perhaps the total number of cells in the buffer as a whole, is not a universal of language, but in fact varies from individual to individual, and perhaps varies consistently from speakers of one language to speakers of another. Thus, a given language, say English, might require, for reasonably full comprehension of spoken language, that grammar rules must be allowed to examine up to three buffer cells. Some speakers of English, however, for unknown reasons, may develop the ability to examine four, or perhaps even five cells at a time. These individuals can thus diagnose any local structural ambiguity which can be resolved within a "window" of that size.

Such an explanation nicely accounts for why half of those asked to read the sentence (vi) above could correctly diagnose its structure, while only some small percentage of those asked to read (vii) could do so. As the reader can verify for himself, (vi) can be accurately diagnosed by a grammar rule that can examine *four* buffer cells simultaneously, while resolving the structure of (vii) requires a rule that examines *five* buffer cells. The results presented above imply that half of those who participated in these experiments can access only three buffer cells at a time, most of the remainder can access four cells, and a few individuals can access up to five cells. (This, of course, implies that it should be the same few people who correctly analyze potential garden paths in each of these experiments. This seems to be exactly the case.)

As a final note, the sentences which I typically asked people to examine were examples which specifically embodied structural ambiguities for which I could not find a fool-proof diagnostic rule. Almost without exception, if I could not find a diagnostic rule which decided between structural analyses A and B for sentences embodying a given ambiguity, at least half the people who were shown example sentences would attempt to analyze all such sentences as if they were examples of structure A, i.e. they took the garden path on all sentences which were examples of structure B, Thus, this theory of garden paths seems to have fairly good predictive value, as least as measured by informal means such as this.

A Theory of Reanalyzing Garden Path Sentences

While the theory stated above provides an account of how a

deterministic parser might be lead down the garden path, it leaves open the question of how such a parser would recover from its error. This section will briefly sketch a highly speculative theory of how this can be done in the context of a natural language understanding system which uses the parser as a subsystem.

A deterministic parser should be viewed, I believe, as a fast, "stupid" black box. According to this view, the grammar of English is really quite simple (which doesn't imply that *finding* the correct grammar is simple), and it costs very little computationally to syntactically analyze a sentence. Because of this simplicity, such a parser is much too weak a mechanism to be able to recover by itself if it runs into difficulty and cannot fully analyze a sentence.

When functioning as part of a larger scale system, given a garden path sentence as input, a deterministic parser ideally should take the garden path and become "stuck" at the point at which people become conscious that they have been misled. (By "stuck", I mean that no grammar rule in any active packet applies.)

When the parser cannot analyze a given input, it will fail with some portion of the input analyzed in some fashion, correct or incorrect. At this point, some higher level "conscious" grammatical problem solving component comes into play. This component restarts the parser on the remainder of the input after activating a special packet of rules which allow the initiation of grammatical fragments. The parser may be restarted once or twice, say, producing a string of two or three grammatical fragments which correspond to the original utterance. Thus, even though the parser cannot successfully analyze the input into a single coherent grammatical structure, it can be used to reduce the input into several coherent pieces which can be given to a higher level analysis process.

Once the original input has been "chunked" into fragments, the higher level problem solver might then use a set of grammatical heuristics to attempt to discover where the parser initially went astray. (For a discussion of parsing using "clever" backtracking which I believe can easily be reinterpreted to yield a set of such heuristics, see Hill [Hill 72].) The problem solver manipulates the fragments using these heuristics until either some solution is found, or the set of available heuristics is exhausted. I believe that a speaker of a language is aware that this latter process is taking place (although certainly not aware of exactly what the process entails).

While this theory is entirely speculative, it does make clear the notion that recovery from garden path sentences and similar phenomena need not be entirely the task of the parser, that subsidiary "error recovery" mechanisms might be called into play. This is the central point of the above theory, and this is all that is really crucial for the account given here.

VII. Conclusions

Many different topics have been discussed in the preceding pages - among them capturing the linguistic generalizations of English, linguistic universals, and psychological phenomena. All have shared one common property, that significant insights into each topic have followed from the Determinism Hypothesis and the structure of the grammar interpreter. This summary will focus on those features of the grammar interpreter which are key to capturing the results presented in this paper. It will attempt to convey exactly which aspects of the grammar interpreter lie at the heart of each of the results established in this paper. This point of view should convey exactly which aspects of PARSIFAL are unique to this parser, and exactly why they are important.

The central idea behind this research is the Determinism Hypothesis, the thesis that the syntax of any natural language can be parsed by a mechanism which operates "strictly deterministically" in that it does not simulate a nondeterministic machine. I have attempted to argue for this thesis only indirectly, by assuming that the Determinism Hypothesis is true, and then seeing what follows from this assumption. If significant insights into the nature of natural language follow from this assumption, and I believe that I have shown that they do, then these insights provide evidence, albeit indirect evidence, for the Determinism Hypothesis.

In fact, most of the arguments presented are one step removed from the hypothesis itself.

In general, I do not show that the results of this paper follow directly from the Determinism Hypothesis, but rather follow from the structure of the grammar interpreter, whose structure in turn is motivated by the Determinism Hypothesis. The structure of PARSIFAL reflects three general principles that follow from the Determinism Hypothesis in conjunction with an examination of the syntax of natural language: that any strictly deterministic parser:

-must be at least partially data driven, but
-must reflect expectations that follow from the partial structures built up during the parsing process; and
-must have some sort of restricted "look-ahead" facility.

PARSIFAL reflects these principles rather directly. The grammar is made up of pattern/action rules, allowing the parser to be data directed. These rules themselves are clustered in packets, which can be activated and deactivated to reflect global expectations. The grammar interpreter's constituent buffer gives it a small window through which to view the input string, allowing restricted look-ahead.

Of the structures that make up the grammar interpreter, it is the constituent buffer which is most central to the results that are presented in this document. The most important of these results follow from the sorts of grammar operations that the buffer makes feasible; others follow from the limitations that its fixed length imposes. All however, hinge crucially upon the existence of the buffer. This data structure, I submit, is the primary source of the power of the parser.

For example:

-Because the buffer automatically compacts upon the attachment of the constituents that it contains, the parsing of a yes/no question and the related declarative will differ in one rule of grammar, with the key difference restricted to the rule patterns and one line of the rules' actions. The yes/no question rule explicitly states only that the NP in the second buffer cell should be attached as the subject of the clause. Because the buffer will then compact, auxiliary parsing rules that expect the subconstituents of the verb cluster to be contiguous will then apply without need for modification.

-Because the buffer provides a three-constituent window on the structure of a clause, diagnostic rules can be formulated that allow the differential diagnosis of pairs of constructions that would seem to be indistinguishable by a deterministic parser without such a buffer. Thus, evidence that would seem to imply that natural language must be parsed nondeterministically can be explained away.

-Because the buffer can only contain a limited number of constituents, the power of these diagnostic rules is limited, leading to a principled distinction between sentences which are perceived as garden paths and those which are not. This explanation for why certain sentences cause garden paths is consistent with the informal experiments presented in section VI. Furthermore, this theory led to the counter-intuitive prediction that as simple a sentence as "Have the packages delivered tomorrow." would cause a garden path, a prediction which was confirmed by informal experiment.

In short, this one mechanism not only allows the formulation of rules of syntax that elegantly capture linguistic generalizations; it also provides the underpinnings for a psychological theory that seems to have high initial plausibility.

Another important source of power is the use of traces, especially in conjunction with the use of the buffer. Especially important is the fact that a trace can be dropped into the buffer, thereby indicating its underlying position in a factorization of the terminal string without specifying its position in the underlying tree. From this follows:

-A simple formulation of passive which accounts for the phenomenon of "raising". The essence of the passive rule - create a trace, bind it to the subject of the current S, drop it into the buffer - is noteworthy in its simplicity. Again, the availability of the buffer yields a very simple solution to a seemingly complex linguistic phenomenon.

-An explanation for the phenomena which underlie Chomsky's Specified Subject Constraint and Subjacency Principle. It is most interesting that the simple formulation of Passive presented here - perhaps the simplest possible formulation of this rule within the framework of PARSIFAL - behaves exactly as if these constraints were true; i.e. that the formulation presented here, by itself and with no extraneous stipulations, leads to the behavior that these constraints attempt to specify.

Directions for Future Work

There are several important areas for which an account must be given if the model presented in this paper is to begin to resemble a full model of the syntactic recognition of natural language. The following is a list of several topics which have not been discussed in this paper, which must be investigated if this model is to be complete:

-No phenomena which seem to require semantic/syntactic interaction, which I believe include such phenomena as *conjunction*, or *PP attachment* have been discussed. These phenomena await future work; I believe that they will require mechanisms outside of the basic grammar interpreter presented here. It should be noted that Woods does not handle these two phenomena within the framework of his ATN mechanism itself, but rather he posits special purpose mechanisms to deal with them. I share his intuitions.

-I have not discussed *lexical ambiguity*, but rather have focussed on *structural ambiguity*. An ambiguity is structural when two different structures can be built up out of smaller constituents of the same given structure and type; an ambiguity is lexical when one word can serve as various parts of speech. While part of the solution to this very pervasive problem seems to involve notions of *discourse context*, I believe that much of the problem can be solved using the techniques presented in this paper; this problem is currently under investigation.

- I have not discussed the psychological difficulty caused by *center embedded sentences* such as "The rat the cat the dog chased bit ate the cheese.". For a recent theory of this phenomenon which is consistent with the model presented here, see [Cowper 76]. (This paper includes an excellent critique of why any parsing model which operates in a purely top-down, i.e. purely hypothesis driven, manner cannot account for this phenomenon.)

Acknowledgments

This paper is a summary of a Ph.D. thesis by the same title. I would like to express my gratitude to the many people who contributed to the technical content of this work: Jon Allen, my thesis advisor, to whom I owe a special debt of thanks, Ira Goldstein, Seymour Papert, Bill Martin, Bob Moore, Chuck Rieger, Mike Genesereth, Gerry Sussman, Mike Brady, Craig Thiersch, Beth Levin, Candy Bullwinkle, Kurt VanLehn, Dave McDonald, and Chuck Rich. A special thanks to Beth Levin for her editorial comments on this paper.

This paper describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the laboratory's artificial intelligence research is provided in part by the Advanced Research Projects Agency of the Department of Defence under Office of Naval Research Contract N00014-75-C-0643.

BIBLIOGRAPHY

Akmajian, A. and F. Heny [1975] *An Introduction to the Principles of Transformational Syntax*, MIT Press, Cambridge, Mass.

Bever, T. G. [1970] "The Cognitive Basis for Linguistic Structures", in J. R. Hayes, ed., *Cognition and the Development of Language*, Wiley and Sons, N.Y.

Bresnan, J. W. [1976] "Evidence for a Theory of Unbounded Transformations", *Linguistic Analysis* 2:353.

Chomsky, N. [1973] "Conditions on Transformations", in S. Anderson and P. Kiparsky, eds., *A Festschrift for Morris Halle*, Holt, Rinehart and Winston, N.Y.

Chomsky, N. [1975] *Reflections on Language*, Pantheon, N.Y..

Chomsky, N. [1976] "Conditions on Rules of Grammar", *Linguistic Analysis* 2:303.

Chomsky, N. [1977] "On Wh-Movement", in A. Akmajian, P. Culicover, and T. Wasow, eds., *Formal Syntax*, Academic Press, N.Y.

Cowper, E. A. [1976] *Constraints on Sentence Complexity: A Model for Syntactic Processing*, unpublished Ph.D. thesis, Brown University.

Fillmore, C. J. [1968] "The Case for Case" in *Universals in Linguistic Theory*, E. Bach and R. T. Harms, eds., Holt, Rinehart, and Winston, N.Y.

Goldstein, I. P. and R. B. Roberts [1977] "NUDGE, A Knowledge-Based Scheduling Program," Memo 405, MIT Artificial Intelligence Laboratory,

Cambridge, Mass.

Gruber, J. S. [1965] *Studies in Lexical Relations*, unpublished Ph.D. thesis, MIT.

Hill, J. M. [1972] *Backup Strategies for Resolving Ambiguity in Natural Language Processing*, unpublished Masters thesis, MIT.

Jackendoff, R. S. [1972] *Semantic Interpretation in Generative Grammar*, MIT Press, Cambridge, Mass..

Marcus, M. P. [1977] *A Theory of Syntactic Recognition for Natural Language*, unpublished Ph.D. thesis, MIT.

Newell, A. and H.A. Simon [1972] *Human Problem Solving*, Prentice-Hall, Englewood Cliffs, N.J.

Postal, P. M. [1974] *On Raising*, MIT Press, Cambridge, Mass.

Pratt, V. R. [1973] "Top-Down Operator Precedence", in the proceedings of *The SIGACT/SIGPLAN Symposium on Principles of Programming Languages*, Boston, Mass.

Ross, J. R. [1967] *Constraints on Variables in Syntax*, unpublished Ph.D. thesis, MIT. Excerpted in G. Harmon, ed. [1974] *On Noam Chomsky*, Anchor, N.Y.

Winograd, T. [1971] *Procedures as a Representation for Data in a Computer Program for Understanding Natural Language*, Project MAC-TR 84, MIT, Cambridge, Mass.

Woods, W. A. [1970] "Transition Network Grammars for Natural Language Analysis", *Communications of the ACM* 13:591.